HC12A4EVBUM/D

May 1996

M68HC12A4EVB

EVALUATION BOARD

USER'S MANUAL

© MOTOROLA Inc., 1996; All Rights Reserved

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

CONTENTS

CHAPTER 1 GENERAL INFORMATION

1.1 INTRODUCTION 1-1
1.2 GENERAL DESCRIPTION AND FEATURES 1-1
1.3 PERFORMANCE NOTES 1-4
1.4 FUNCTIONAL OVERVIEW 1-5
1.5 EXTERNAL EQUPMENT REQUIREMENTS 1-6
1.6 EVB SPECIFICATIONS 1-7
1.7 CUSTOMER SUPPORT 1-9

CHAPTER 2 CONFIGURATION AND SETUP

2.1 UNPACKING AND PREPARATION 2-1
2.2 EVB CONFIGURATION 2-2
2.3 EVB TO POWER SUPPLY CONNECTION 2-2
2.4 EVB TO TERMINAL CONNECTION 2-3
2.5 TERMINAL COMMUNICATIONS SETUP 2-4
2.5.1 Communication Parameters 2-4
2.5.2 Dumb-Terminal Setup 2-5
2.5.3 Host-Computer Setup 2-5
2.5.4 Changing the Baud Rate 2-5
2.6 USING FAST EXTERNAL RAM 2-6
2.6.1 Selecting and Replacing the RAM Chips 2-6
2.6.2 Reprogramming the RAM Chip Select 2-6

CHAPTER 3 OPERATION

3.1 STARTUP 3-1
3.2 RESET 3-2
3.3 PROGRAM ABORT 3-2
3.4 USING D-BUG12 COMMANDS 3-3

CHAPTER 3 OPERATION (continued)

3.5 D-BUG12 COMMAND SET 3-5
Assembler/Disassembler 3-6
Set Baud Rate 3-9
Block Fill 3-10
Breakpoint Set 3-11
Bulk Erase On-Chip EEPROM 3-12
Call Subroutine 3-13
Go Execute a User Program 3-14
Go Till 3-15
Onscreen Help Summary 3-16
Load S-Record File 3-17
Memory Display 3-18
Memory Display, Word 3-19
Memory Modify 3-20
Memory Modify, Word 3-21
Move Memory Block 3-22
Remove Breakpoints 3-23
Register Display 3-24
Register Modify 3-25
Trace 3-26
Display Memory in S-Record Format 3-28
Verify S-Record File against Memory 3-29
Modify Register Value 3-30
3.6 ALTERNATE EXECUTION FROM EEPROM 3-32
3.7 OFF-BOARD CODE GENERATION 3-32
3.8 MEMORY USAGE 3-33
3.8.1 Description 3-33
3.8.2 Memory Map 3-34
3.9 OPERATIONAL LIMITATIONS 3-34
3.9.1 On-Chip RAM 3-34
3.9.2 SCI Port Usage 3-35
3.9.3 Dedicated MCU Pins 3-35
3.9.4 Terminal Communications 3-35

CHAPTER 4 HARDWARE REFERENCE

4.1 PCB DESCRIPTION 4-1
4.2 CONFIGURATION HEADERS AND JUMPER SETTINGS 4-1
4.3 POWER INPUT CIRCUITRY 4-6
4.4 TERMINAL INTERFACE 4-6
4.5 MICROCONTROLLER 4-7
4.6 MEMORY 4-9
4.6.1 Memory Types and Sockets 4-9
4.6.2 Chip Selects 4-11
4.6.3 Glue Logic 4-12
4.7 CLOCK CIRCUITRY 4-13
4.8 PHASE-LOCKED LOOP (PLL) 4-14
4.9 RESET 4-14
4.10 LOW-VOLTAGE INHIBIT 4-14
4.11 ANALOG-TO-DIGITAL (A/D) CONVERTER 4-14
4.12 BACKGROUND DEBUG MODE (BDM) INTERFACE 4-15
4.13 PROTOTYPE AREA 4-15
4.14 MCU CONNECTORS 4-17

APPENDIX A S-RECORD FORMAT

DESCRIPTION A-1
S-RECORD CONTENT A-1
S-RECORD TYPES A-2
S-RECORD EXAMPLE A-3

APPENDIX B COMMUNICATIONS PROGRAM EXAMPLES

INTRODUCTION B-1
PROCOMM FOR DOS — IBM PC B-1
Setup B-1
S-Record Transfers to EVB Memory B-2
KERMIT FOR DOS — IBM PC B-3
Setup B-3
S-Record Transfers to EVB Memory B-3

APPENDIX B COMMUNICATIONS PROGRAM EXAMPLES (continued)

KERMIT — SUN WORKSTATION....B-4
Setup....B-4
S-Record Transfers to EVB Memory....B-4
MACTERMINAL — APPLE MACINTOSH....B-5
Setup....B-5
S-Record Transfers to EVB Memory....B-5
RED RYDER — APPLE MACINTOSH....B-6
Setup....B-6
S-Record Transfers to EVB Memory....B-6

APPENDIX C D-BUG12 STARTUP CODE

APPENDIX D D-BUG12 CUSTOMIZATION DATA

APPENDIX E CUSTOMIZING THE EPROMS

APPENDIX F SDI CONFIGURATION

INDEX

FIGURES

1-1. EVB Layout and Component Placement 1-4
1-2. System Block Diagram 1-5
2-1. EVB Power Connector J6 2-3
4-1. Memory Sockets Configuration 4-10
4-2. Chip Select Header 4-12
4-3. RAM/ROM Logic Diagram 4-13
4-4. Prototype Area (Component-Side View) 4-16
4-5. MCU Connector J8 (Component-Side View) 4-17
4-6. MCU Connector J9 (Component-Side View) 4-18

TABLES

1-1. EVB Specifications 1-10
2-1. RS-232C Interface Cabling 2-4
2-2. Communication Parameters 2-5
3-1. D-Bug12 Command-Set Summary 3-4
3-2. M68HC11 to CPU12 Instruction Translation 3-7
3-3. CPU12 Registers 3-30
3-4. Condition Code Register Bits 3-31
3-5. Factory-Configuration Memory Map 3-34
4-1. Jumper-Selectable Functions 4-3
4-2. CPU Mode Selection 4-8
4-3. EVB Memories Supplied 4-11
4-4. BDM Connector J5 Pin Assignments 4-15
4-5. MCU Connector J8 Pin Assignments 4-19
4-6. MCU Connector J9 Pin Assignments 4-21

CHAPTER 1
GENERAL INFORMATION

1.1 INTRODUCTION

This manual provides the necessary information for using the M68HC12A4EVB Evaluation Board (the EVB), an evaluation, debugging, and code-generation tool for the MC68HC812A4 Microcontroller Unit (MCU) devices. The manual includes:

- A general description of the EVB
- Configuration and setup instructions
- Startup and operating instructions
- Detailed descriptions of the operating firmware's command set
- A detailed hardware-reference section
- Appendices containing reference data

Additional reference items, such as schematic diagrams and parts lists, are shipped as part of the EVB package.

1.2 GENERAL DESCRIPTION AND FEATURES

The EVB is an economical tool for designing and debugging code for, and evaluating the operation of, the MC68HC12 MCU family. By providing the essential MCU timing and I/O circuitry, the EVB simplifies user evaluation of prototype hardware and software.

The board consists of an 8-inch by 8-inch multi-layer printed circuit board (PCB) that provides the platform for interface and power connections to the MC68HC812A4 MCU chip, which is installed in a production socket.

Figure 1-1 shows the EVB's layout and locations of the major components, as viewed from the component side of the board.

The block diagram in Figure 1-2 depicts the logical relationships and interconnections within the EVB and with external equipment.

Hardware features of the EVB include:

- Power, ground, and 4 signal planes
- Single-supply +3 to +5 Vdc power input (J6)

- Two RS-232C interfaces
- Two memory sockets populated with two 32Kx8 EPROMs (U7, U9A), containing the D-Bug12 monitor program
- Two memory sockets populated with two 8Kx8 SRAMs (U4, U6A)
- Support for up to 1 MB of program space and 512 KB of data space using optional memory configurations
- 16-MHz crystal-controlled clock oscillator (Y2) in a socket that can accommodate optional 8- or 14-pin oscillator chips (XY2)
- Headers for jumper selection of hardware options:(1)
  - Low-voltage inhibit (W1)
  - RAM write-protection (W3)
  - MCU chip selects for memory devices (W11)
  - RAM function select (W12, W13)
  - ROM function select (W22, W24, W29, W32, W33, W36)
  - MCU mode control (W30, W34, W42)
  - Alternate execution from on-chip EEPROM (W20)
  - Serial Communications Interface (SCI) configuration (W10, W14, W21)
- Two 2x30 header connectors for access to the MCU's I/O and bus lines (J8 and J9)
- Prototype expansion area for customized interfacing with the MCU
- Low-profile reset (S1) and program-abort (S2) push-button switches
- Low voltage inhibit protection (U1)
- LED power-on indicator (DS1)
- Test points for ground connections around the board (E1, E2, E3, E12, E13, E14)
- 2x3 header (J5) provides a connector for using background debug development tools such as the Serial Debug Interface (SDI)
- Phase-Locked Loop (PLL) biasing circuitry for altering the MCU's time base

(1)For full details of the jumper settings, refer to Table 4-1.

Firmware features include:

- The D-Bug12 monitor/debugger program, resident in external EPROM
- Full support for either dumb-terminal or host-computer terminal interface
- Single-line assembler/disassembler
- File-transfer capability from a host computer, allowing off-board code generation

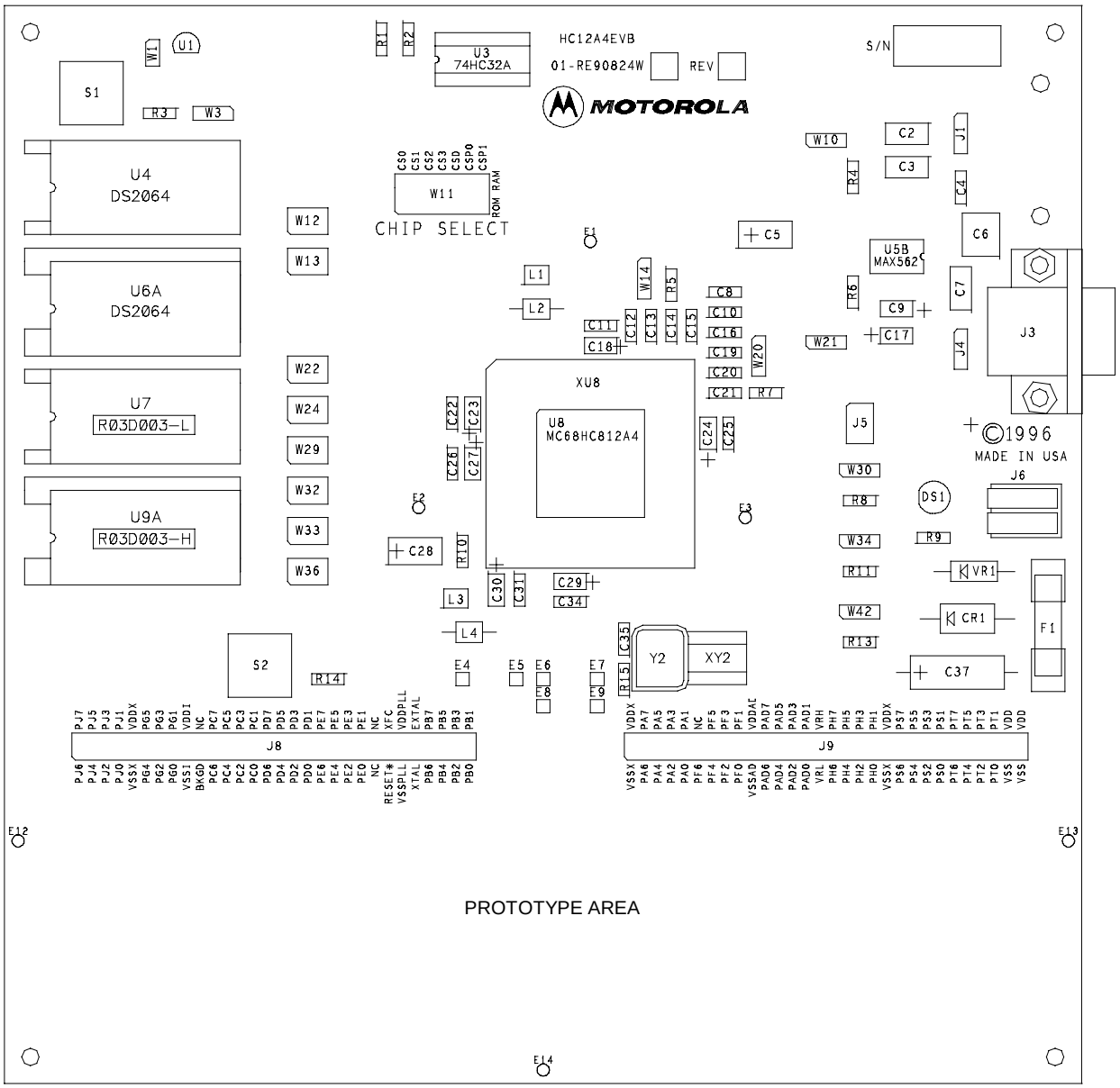


Figure 1-1. EVB Layout and Component Placement

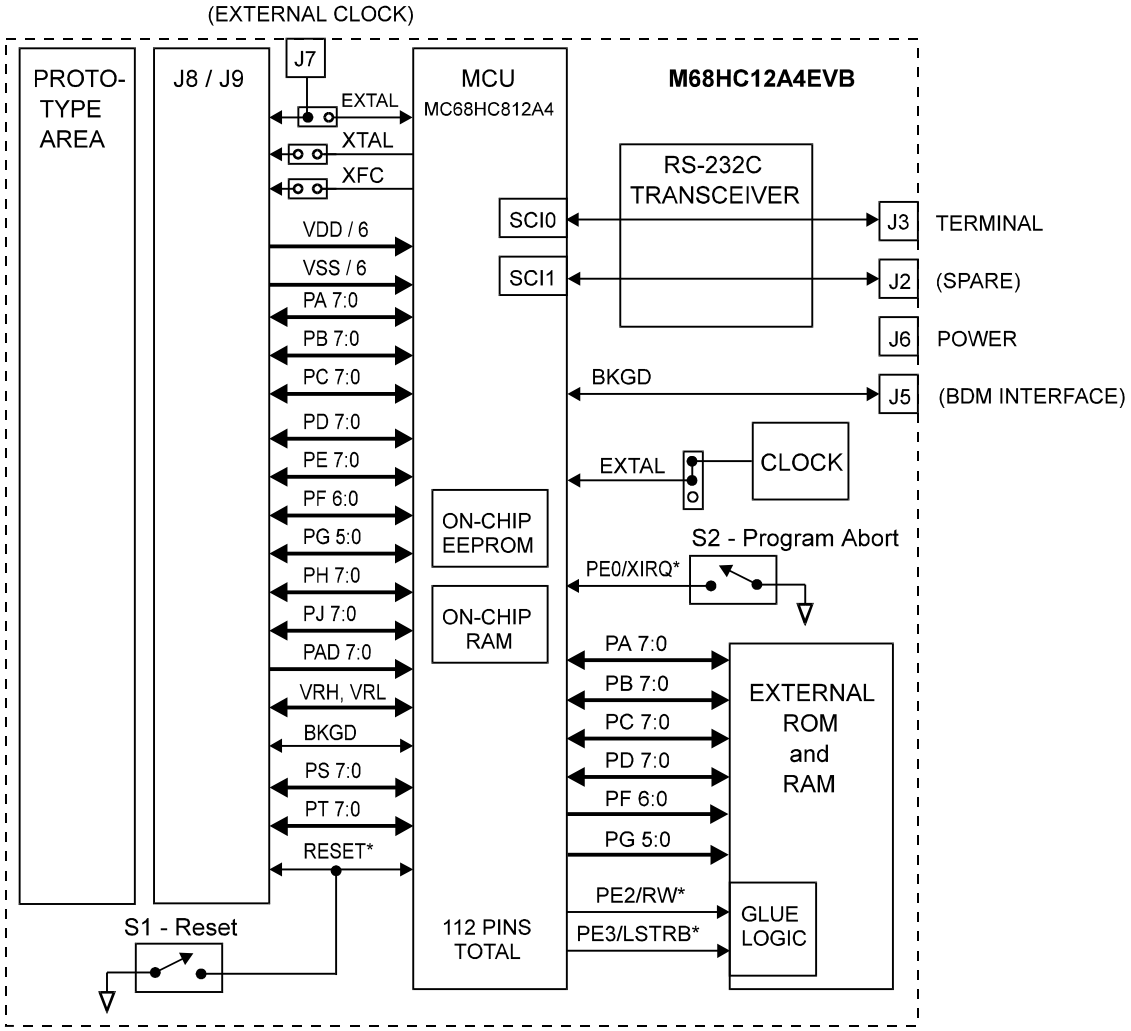


Figure 1-2. System Block Diagram

1.3 PERFORMANCE NOTES

The M68HC12A4EVB's external RAM memory chips, U4 and U6A, were chosen to emphasize the EVB's low-voltage and low-power operational capability over the range of +3.5 to +5.0 Vdc.

However, these parts are not fast enough to operate at the 16-MHz speed of the factory-supplied clock oscillator. In order to use them at this external clock speed, the D-Bug12 startup code programs the MCU's RAM chip select to insert one "wait state" into each access of external RAM. Thus, when programs are run from external RAM, performance is approximately 40% slower than it would be if the RAM chips were fast enough to run without wait states. Typical software performance improvements of 80% - 95% can be realized with faster external RAM.

For high-speed performance, the factory-supplied RAM devices may be replaced with faster parts that allow programs to execute at the full external clock speed. Two steps are required for this:

1. Replace the RAM devices, U4 and U6A, with faster parts.
2. Modify the RAM chip select to eliminate the wait state (E-clock stretch).

Detailed instructions for these procedures are found in **2.6 Using Fast External RAM**.

NOTES

Programs that execute exclusively from the MCU's on-chip RAM and EEPROM always run at the full clock speed. No wait states are introduced when accessing these areas.

Table 3-5, the default memory map, depicts the addresses of the EVB's different memory areas.

1.4 FUNCTIONAL OVERVIEW

The EVB is factory-configured to execute D-Bug12, the EPROM-resident monitor program, without further configuration by the user. It is ready for use with an RS-232C terminal for writing and debugging user code. Follow the setup instructions in Chapter 2 to prepare for operation.

Optionally, the EVB can accommodate various types and configurations of external memory to suit a particular application's requirements. These custom configurations are effected by installing the appropriate memory chips in the EVB's memory sockets and by setting jumpers on the EVB to correctly establish the MC68HC812A4's memory-access operations. Table 1-1 lists the allowable sizes and types of memory. For the correct jumper settings, refer to **4.2 Configuration Headers and Jumper Settings**.

NOTE

The D-Bug12 operating instructions in this manual presume the factory-default memory configuration. Other configurations require different operating-software arrangements.

The MC68HC812A4's two Serial Communications Interface (SCI) ports are associated with separate RS-232C interfaces. D-Bug12 uses one of the SCIs for communications with the user terminal (jumper-selectable; SCI0 by default). The second SCI port is available for user applications. For information on the ports and their connectors, refer to **2.4 EVB to Terminal Connection** and **4.4 Terminal Interface**.

If the MCU's single-wire background debug mode (BDM) interface serves as the user interface, both of the SCI ports become available for user applications. This mode requires a background debug development tool, such as Motorola's Serial Debug Interface (SDI), and a host computer

with the appropriate interface software. For more information, refer to Appendix F and to the *Motorola Serial Debug Interface User's Manual.*

NOTE

D-Bug12 does not use the BDM interface.

Two methods may be used to generate EVB user code:

1. For small programs or subroutines, D-Bug12's single-line assembler/disassembler may be used to place object code directly into the EVB's memory.
2. For larger programs, the Motorola MCUasm assembler may be used on a host computer to generate S-Record object files, which can then be loaded into the EVB's memory using D-Bug12's LOAD command.

The EVB features a prototype area, which allows custom interfacing with the MCU's I/O and bus lines. These connections are broken out via headers J8 and J9, which are immediately adjacent to the prototype area as shown in Figure 1-1.

An on-board push-button switch, S1, provides for resetting the EVB hardware and restarting D-Bug12. Another on-board switch, S2, allows aborting the execution of a user program — useful in regaining control of a runaway program. Both of these switch functions are available for customized use in the prototype area.

The EVB can begin operation in either of two jumper-selectable (W20) modes at reset. In normal mode, D-Bug12 immediately issues its command prompt on the terminal display and waits for a user entry. In the alternate mode, execution begins directly with the user code in on-chip EEPROM. This hardware function is also available for customized use in the prototype area.

D-Bug12 allows programming of the MC68HC812A4's on-chip EEPROM through commands that directly alter memory. For full details of all the commands, refer to **3.5 D-Bug12 Command Set**.

Due to the fact that the MCU must manage the EVB hardware and execute D-Bug12 *in addition* to serving as the user-application processor, there are a few restrictions on its use. For more information, refer to **3.9 Operational Limitations**.

1.5 EXTERNAL EQUPMENT REQUIREMENTS

In addition to the EVB, the following user-supplied external equipment is required:

- Power supply — see Table 1-1 for voltage and current requirements.

NOTE

Table 1-1 indicates that EVB operation at +3.0 Vdc requires the slower clock speed of 8 MHz. This limitation applies to programs (including the operating firmware, D-Bug12) that use external memory.

If an application program uses on-chip RAM and EEPROM *exclusively* — i.e., if external memory is not used — the clock speed can remain at 16 MHz with a supply voltage of +3.0 Vdc.

- User terminal — options:
  - RS-232C dumb terminal — allows single-line on-board code assembly and disassembly.
  - Host computer with RS-232C serial port — allows off-board code assembly that can be loaded into the EVB's memory. Requires a user-supplied communications program capable of emulating a dumb terminal. Examples of acceptable communications programs are given in Appendix B.
  - Host computer using the MCU's BDM interface — frees both of the MCU's SCI ports for user applications. Requires a background debug development tool, such as the Motorola Serial Debug Interface (SDI), and the appropriate interface software.
- Power-supply and terminal interconnection cables as required

For full details of equipment setup, cabling, and special requirements, refer to Chapter 2.

1.6 EVB SPECIFICATIONS

Table 1-1 lists the EVB specifications.

Table 1-1. EVB Specifications

Characteristic	Specifications
MCU	MC68HC812A4
SRAM maximum memory: Wide mode Narrow mode	 16, 64, 256, or 1024 Kbytes 8, 32, 128, or 512 Kbytes
ROM maximum memory: EPROM: Wide mode Narrow mode EEPROM: Wide mode Narrow mode	 64, 128, 256, 512, or 1024 Kbytes 32, 64, 128, 256, or 512 Kbytes 64, 128, 256, or 512 Kbytes 32, 64, 128, or 256 Kbytes
MCU I/O ports	HCMOS compatible
Background Debug Mode interface	2x3 header
Communications ports	Two RS-232C DCE ports
Power requirements: 16 MHz clock source 8 MHz clock source	 +3.5 to +5.0 Vdc @ 150 mA (max.), fuse-protected @ 1.5 A +3.0 to +5.0 Vdc @ 150 mA (max.), fuse-protected @ 1.5 A
Prototype area: Area Holes	 2 x 8 in. (approx.) 79 wide x 20 high (0.1 in. centers)
Board dimensions	8 x 8 in.

1.7 CUSTOMER SUPPORT

AUSTRALIA,
- Melbourne – (61-3)887-0711
- Sydney – (61-2)906-3855

BRAZIL
- Sao Paulo – 55(11)815-4200

CANADA
- B.C., Vancouver – (604)293-7650
- ONTARIO, Toronto – (416)497-8181
- ONTARIO, Ottawa – (613)226-3491
- QUEBEC, Montreal – (514)333-3300

CHINA
- Beijing – 86-505-2180

FINLAND
- Helsinki – 358-0-351 61191

FRANCE
- Paris – 33134 635900

GERMANY
- Langenhagen/Hannover – 49(511)786880
- Munich – 49 89 92103-0
- Nuremberg – 49 911 96-3190
- Sindelfingen – 49 7031 79 710
- Wiesbaden – 49 611 973050

HONG KONG
- Kwai Fong – 852-6106888
- Tai Po – 852-6668333

INDIA
- Bangalore – (91-812)627094

ISRAEL
- Herzlia – 972-9-590222

ITALY
- Milan – 39(2)82201

JAPAN
- Fukuoka – 81-92-725-7583
- Gotanda – 81-3-5487-8311

JAPAN
- Nagoya – 81-52-232-3500
- Osaka – 81-6-305-1802
- Sendai – 81-22-268-4333
- Takamatsu – 81-878-37-9972
- Tokyo – 81-3-3440-3311

KOREA
- Pusan – 82(51)4635-035
- Seoul – 82(2)554-5118

MALAYSIA
- Penang – 60(4)374514

MEXICO
- Mexico City – 52(5)282-0230
- Guadalajara – 52(36)21-8977

NETHERLANDS
- Best – (31)4998 612 11

PUERTO RICO
- San Juan – (809)793-2170

SINGAPORE – (65)4818188

SPAIN
- Madrid – 34(1)457-8204

SWEDEN
- Solna – 46(8)734-8800

SWITZERLAND
- Geneva – 41(22)799 11 11
- Zurich – 41(1)730-4074

TAIWAN
- Taipei – 886(2)717-7089

THAILAND
- Bangkok – 66(2)254-4910

UNITED KINGDOM
- Aylesbury – 44(296)395-252

UNITED STATES
- Phoenix, AZ – 1-800-441-2447

For a list of the Motorola sales offices and distributors: http://freeware.aus.sps.mot.com/

CHAPTER 2
CONFIGURATION AND SETUP

2.1 UNPACKING AND PREPARATION

Verify that the following items are present in the EVB package:

- The M68HC12A4EVB board assembly
- Warranty and registration cards
- EVB schematic diagram and parts list
- *M68HC12A4EVB User's Manual*
- *MC68HC812A4 Technical Summary*
- *CPU12 Reference Manual*
- *MC68HC12 Family Brochure*
- Demo software
- Assembly Language Development Toolset
- *Using D-Bug12 Callable Routines*

Save all packing materials for storing and shipping the EVB.

Remove the EVB from its anti-static shipping bag.

Carefully remove the protective case and conductive foam that cover the MCU and its socket during shipment.

Inspect the alignment of the MCU's pins within its socket. If it appears necessary to reseat the MCU,

1. press down on two opposite sides of the MCU socket
2. gently press the MCU chip into place
3. release the MCU socket.

Verify that all other socketed parts are correctly seated.

2.2 EVB CONFIGURATION

Because the EVB has been factory-configured to operate with D-Bug12, it is not necessary to change any of the jumper settings to begin operating immediately.

Only one jumper (header W20) should be changed during the course of factory-default EVB operation with D-Bug12:

pins 2-3 jumpered (default) — Normal execution mode. D-Bug12 is executed from external EPROM upon reset. The D-Bug12 prompt appears immediately on the terminal display.

pins 1-2 jumpered — Alternate execution mode. User code is executed from on-chip EEPROM upon reset. For more information, refer to **3.6 Alternate Execution from EEPROM**.

Other jumper settings affect the hardware setup and/or MCU operational modes. For an overview of all jumper-selectable functions, refer to **1.2 General Description and Features**. For details of the settings, see Table 4-1.

2.3 EVB TO POWER SUPPLY CONNECTION

The EVB requires a user-provided external power supply. See Table 1-1 for the voltage and current specifications. For full details of the EVB's power-input circuitry, refer to **4.3 Power Input Circuitry**.

Although fuse protection is built into the EVB, a power supply with current-limiting capability is desirable. If this feature is available on the power supply, set it to 200 mA.

Connect the external power supply to connector J6 on the EVB as shown in Figure 2-1, using 20 AWG or smaller insulated wire. Strip each wire's insulation 1/4 in. from the end, lift the J6 contact lever to release tension on the contact, insert the bare end of the wire into J6, and close the lever to secure the wire. Observe the polarity carefully.

CAUTION

Do not use wire larger than 20 AWG in connector J6. Larger wire could damage the connector.

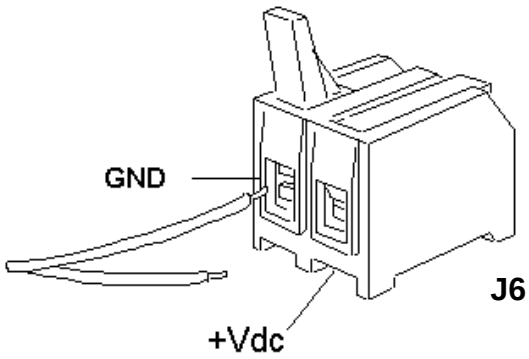


Figure 2-1. EVB Power Connector J6

2.4 EVB TO TERMINAL CONNECTION

For factory-default operation, connect the terminal to J3 or J4 on the EVB, as shown in Table 2-1. This setup uses the MCU's SCI port 0 (SCI0) and its associated RS-232C interface for communications with the terminal device.

To use SCI1 and the second RS-232C interface for the terminal, the EVB's hardware setup must be modified. For details, refer to **4.4 Terminal Interface**.

Standard, commercially available cables may be used in most cases. Note that the EVB uses only three of the RS-232C signals. Table 2-1 lists these signals and their pin assignments.

The EVB's RS-232C connectors, J2 (default) and J3 (unpopulated footprint), are wired as Data Circuit-terminating Equipment (DCE) and employ 9-pin subminiature D (DB-9) receptacles. The equivalent 3-pin headers, J1 and J4, serve the same purposes and may be used for customized cabling.

Most terminal devices — whether dumb terminals or the serial ports on host computers — are wired as Data Terminal Equipment (DTE) and employ 9- or 25-pin subminiature D (DB-9 or DB-25) plugs. In these cases, normal straight-through cabling is used between the EVB and the terminal. Adapters are readily available for connecting 9-pin cables to 25-pin terminal connectors.

If the terminal device is wired as DCE, the RXD and TXD lines must be cross-connected, as shown in Table 2-1. Commercial "null modem" adapter cables are available for this purpose.

Table 2-1. RS-232C Interface Cabling

EVB Pins (always DCE)		DTE Signal	Terminal Pins			
J3[1] / J2[2] DB-9 Receptacle	J4[1] / J1[2] 3-Pin Header		DTE[3] Plug		DCE[4] Receptacle	
			DB-9	DB-25	DB-9	DB-25
2	2	Receive Data (RXD)	2	2	3	3
3	3	Transmit Data (TXD)	3	3	2	2
5	1	Ground (GND)	5	7	5	7

[1] Factory default (terminal interface uses SCI0)
[2] Optional (terminal interface uses SCI1). Hardware modifications are required. For details, refer to **4.4 Terminal Interface**.
[3] Normal (DCE-to-DTE) cable connections
[4] Null modem (DCE-to-DCE) cable connections

Optionally, the MCU's background debug mode (BDM) interface can serve as the user interface. This setup makes both of the SCI ports available for user applications. Additional hardware and software are required. For more information, refer to the documentation for the background debug development tool being used, such as Motorola's Serial Debug Interface.

NOTE

D-Bug12 does not use the BDM interface.

2.5 TERMINAL COMMUNICATIONS SETUP

2.5.1 Communication Parameters

The EVB's serial communications ports use the communication parameters listed in Table 2-2. Of these, only the baud rate can be changed. For instructions on changing it, refer to **2.5.4 Changing the Baud Rate**.

Table 2-2. Communication Parameters

Baud Rate	9600
Data Bits	8
Stop Bits	1
Parity	none

2.5.2 Dumb-Terminal Setup

Configuring a dumb terminal for use with the EVB consists of setting its parameters as shown in Table 2-2. Many terminals are configurable with externally accessible switches, but the procedure differs between brands and models. Consult the manufacturer's instructions for the terminal being used.

2.5.3 Host-Computer Setup

One advantage of using a host computer as the EVB's terminal is the ability to generate code off-board, for subsequent loading into the EVB's memory. It is thus desirable for the host to be capable of running programs such as Motorola's MCUasm assembler. For more information, refer to **3.7 Off-Board Code Generation**.

To serve as the EVB's terminal, the host computer must have an RS-232C serial port and an installed communications program capable of operating with the parameters listed in Table 2-2.

Setting up the parameters is normally done within the communications program, after it has been started on the host. Usually, the setup can be saved in a configuration file so that it does not have to be repeated. Procedures vary between programs; consult the user's guide for the specific program.

Appendix B provides examples of using some of the commonly available communications programs.

2.5.4 Changing the Baud Rate

The EVB's default baud rate for the RS-232C ports is 9600. This can be changed in two ways:

- For temporary changes, use the D-Bug12 BAUD command. This change remains in effect only until the next reset or power-up, at which time the baud rate returns to 9600.
- For permanent changes, the D-Bug12 baud-rate initialization value stored in EPROM must be modified. For instructions, refer to Appendix D and Appendix E.

2.6 USING FAST EXTERNAL RAM

To replace the two factory-supplied SRAM chips with parts capable of operation at the full 16-MHz external clock speed (8-MHz E-clock) with no wait states, two operations are required:

1. Replace the SRAM chips with suitably fast parts — section 2.6.1.
2. Reprogram the SRAM chip select, CSD*, for zero-wait-state operation — section 2.6.2.

2.6.1 Selecting and Replacing the RAM Chips

The replacement 8K x 8 SRAM devices should have a chip-select access time of less than 60 nanoseconds. An example of a device that has been used successfully is the Integrated Device Technologies part number IDT7164L25P (8K x 8, 25 ns.).

When installing the replacement SRAM devices, make sure that their pins align with the rightmost ends of sockets U4 and U6A, as viewed in Figure 1-1.

2.6.2 Reprogramming the RAM Chip Select

Either of two methods may be used to reprogram the RAM chip select, CSD*, to eliminate the wait state.

NOTE

Before attempting either of the following methods, ensure that the EVB is operating properly by following the startup instructions in section 3.1.

Method A — modifying the CSSTR0 register in memory (temporary)

This method may be used without altering the D-Bug12 startup code in EPROM. However, it *must be repeated* each time the EVB is powered up or reset.

Using D-Bug12's MM command, change the value at memory location $003E from $05 to $04.

Method B — modifying the D-Bug12 startup code in EPROM (permanent)

This method is accomplished by reprogramming a single byte in the factory-supplied, one-time-programmable (OTP) EPROM, U7. An EPROM programmer is required.

NOTES

This method *does not work in reverse*. If U7 has already been reprogrammed using this technique, it *cannot be restored* to its original state.

If the EPROMs are to be customized in some other way — for example, to add a user program or to modify another aspect of D-Bug12 — the change to register CSSTR0 can be made in the startup source code. For more information, refer to Appendix C, D-Bug12 Startup Code, and Appendix E, Customizing the EPROMs.

To permanently reprogram U7 for zero RAM wait states, follow these steps:

1. Remove power from the EVB.
2. Being careful not to bend any pins, remove U7 from its socket on the EVB and install it in the appropriate socket on the EPROM programmer.
3. Following the instructions and using the software for the EPROM programmer, perform the steps in ***Procedure 1*** or ***Procedure 2***, as described below.

Some EPROM programmers do not have an editable RAM buffer capable of holding the entire contents of U7. Instead, they program EPROMs directly from the contents of a disk file.

If the programmer being used has an editable RAM buffer large enough to hold the contents of U7, use ***Procedure 1***. Otherwise, to reprogram U7 from a disk file, use ***Procedure 2***.

Procedure 1

1. Select the Atmel device type AT27LV256R.
2. Read the contents of U7 into the EPROM programmer's editable RAM buffer.
3. Before modifying U7, save a copy of its contents to a disk file for backup purposes.
4. Change the contents of the programmer's editable RAM buffer at location $7ED6 from $05 to $ 04.
5. Reprogram U7 with the edited contents of the programmer's RAM buffer.
6. Reinstall U7 in its socket on the EVB. Be sure that its pins align with the rightmost end of its socket, as viewed in Figure 1-1.
7. Apply power to the EVB and press S1, the reset switch. The D-Bug12 prompt should appear on the terminal display.
8. Ensure that the modification was performed properly by using D-Bug12's MD command to examine the CSSTR0 register at memory location $003E. It should contain the value $04.

Procedure 2

1. Create a text file containing the following two lines:

   ```
   S1047E6D040C
   S9030000FC
   ```

2. Select the Atmel device type AT27LV256R.
3. Before modifying U7, save a copy of its contents to a disk file for backup purposes.
4. Reprogram U7 with the contents of the text file created in Step 1.
5. Reinstall U7 in its socket on the EVB. Be sure that its pins align with the rightmost end of its socket, as viewed in Figure 1-1.
6. Apply power to the EVB and press S1, the reset switch. The D-Bug12 prompt should appear on the terminal display.
7. Ensure that the modification was performed properly by using D-Bug12's MD command to examine the CSSTR0 register at memory location $003E. It should contain the value $04.

CHAPTER 3
OPERATION

3.1 STARTUP

The following startup procedure includes a checklist of configuration and setup items from Chapter 2. To begin operating the M68HC12A4EVB, follow these steps:

1. Configure the EVB if required — section 2.2.
2. Determine whether execution should begin with the D-Bug12 monitor program (factory default) or with user code in on-chip EEPROM. Set the jumper on header W20 accordingly — sections 2.2 and 3.6.
3. Connect the EVB to the external power supply — section 2.3.
4. Connect the EVB to the terminal — section 2.4.
5. Configure the terminal communications interface — section 2.5.
6. Apply power to the EVB and to the terminal. If the terminal is a host computer,
   a. Verify that it has booted correctly.
   b. Start the communications program for terminal emulation — section 2.5.3 and Appendix B.
7. Reset the EVB by pressing and releasing the on-board reset switch (S1).

If the EVB is configured to execute D-Bug12 upon reset (factory default — startup step 2), the D-Bug12 sign-on banner and prompt should appear on the terminal's display as follows:

```
D-Bug12 v1.0.2
Copyright 1995 - 1996 Motorola Semiconductor
For Commands type "Help"
>
```

If the prompt does not appear, check all connections and verify that startup steps 1 through 7 above have been performed correctly.

When the prompt appears, D-Bug12 is ready to accept commands from the terminal as described in sections 3.4 and 3.5.

If the EVB is configured to execute user code upon reset (startup step 2), the code in on-chip EEPROM is executed immediately. For more information, refer to **3.6 Alternate Execution from EEPROM**. Control can be returned to the D-Bug12 terminal prompt by doing one of the following:

1. Terminating the user code with appropriate instructions — see section 3.6.
2. Activating the program-abort function — see section 3.3.

3.2 RESET

EVB operation can be restarted at any time by activating the hardware reset function. Do this in one of two ways:

1. Press and release the on-board reset switch, S1 (always applicable).
2. If the hardware reset input has been customized in the prototype area, activate it in accordance with the custom circuitry.

Note that the EVB's reset circuitry is associated with the low-voltage inhibit protection. For more information, refer to **4.9 Reset** and **4.10 Low-Voltage Inhibit**.

3.3 PROGRAM ABORT

During software development, bugs in the code can cause a program to get stuck in an endless loop, thereby preventing proper response (i.e., a "crash"). In these situations, use the EVB's program-abort function to return control of execution to D-Bug12, which then displays the register contents at the point where the user program was terminated.

Activating the program-abort function asserts the MCU's XIRQ* hardware interrupt line. There are restrictions on its use under certain circumstances; refer to **3.9 Operational Limitations**.

Activate the program-abort function by doing one of the following:

1. Press and release the on-board program-abort switch, S2.
2. If the program-abort input has been customized in the prototype area, activate it in accordance with the custom circuitry.

NOTE

If the EVB is configured to begin execution from on-chip EEPROM, D-Bug12 jumps to the starting EEPROM address without before performing all of its initialization and is thus not operable. *Do not* activate the program-abort function under these conditions. Instead, move the jumper on header W20 to pins 2-3 and activate the reset function to return control to D-Bug12.

3.4 USING D-BUG12 COMMANDS

D-Bug12, the EVB's firmware-resident monitor program, provides a self-contained operating environment that allows writing, evaluation, and debugging of user programs.

Commands are typed on the terminal's D-Bug12 prompt line and executed when the carriage-return (ENTER) key is pressed. D-Bug12 then displays either the appropriate response to the command or an error indication.

The D-Bug12 command-line prompt is the greater-than sign (>). Type the command and any other required or optional fields immediately after the prompt, as follows:

command-line syntax:

```
<command>  [<parameter>]  ...[<parameter>]<ENTER>
```

where:

<command>	is the command mnemonic.
<parameter>	is an expression or address.
<ENTER>	is the terminal keyboard's carriage-return or enter key.

NOTES

1. The command-line syntax is illustrated using the following special characters for clarification. *Do not* type these characters on the command line:

< >	required syntactical element
[ ]	optional field
...[ ]	repeated optional fields

2. Fields are separated by any number of space characters.
3. All numeric fields, unless noted otherwise, are interpreted as hexadecimal.
4. Command-line entries are case-insensitive and may be typed using any combination of upper- and lower-case letters.

5. A maximum of 80 characters, including the terminating carriage return, may be entered on the command line. After the 80th character, D-Bug12 automatically terminates the command-line entry and processes the characters entered to that point.
6. Before the <ENTER> key is pressed, the command line may be edited using the backspace key. Receiving the backspace character causes D-Bug12 to delete the previously-received character from its input buffer and erase the character from the display.

Table 3-1 summarizes the D-Bug12 commands. For detailed descriptions of each command, refer to **3.5 D-Bug12 Command Set**.

Table 3-1. D-Bug12 Command-Set Summary

Command	Description
ASM <address>	Single-line assembler/disassembler
BAUD <BAUDRate>	Set the SCI communications baud rate
BF <StartAddress><EndAddress> [<Data>]	Block Fill user memory with data
BR [<Address><Address>...]	Set/display user breakpoints
BULK	Bulk erase on-chip EEPROM
CALL [<Address>]	Execute a user subroutine; return to D-Bug12 when finished
G [<Address>]	Go — begin execution of user program
GT <Address>	Go Till — set a temporary breakpoint and begin execution of user program
HELP	Display D-Bug12 command set and command syntax
LOAD [<AddressOffset>]	Load user program in S-Record format*
MD <StartAddress> [<EndAddress>]	Memory Display — display memory contents in hex bytes/ASCII format
MDW <StartAddress> [<EndAddress>]	Memory Display Word — display memory contents in hex words/ASCII format
MM <Address> [<data>]	Memory Modify — interactively examine/change memory contents
MMW <address> [<data>]	Memory Modify Word — interactively examine/change memory contents
MOVE <StartAddress> <EndAddress> <DestAddress>	Move a block of memory
NOBR [<Address> <Address>...]	Remove individual user breakpoints

Table 3-1. D-Bug12 Command-Set Summary (continued)

Command	Description
RD	Register Display — display the CPU register contents
RM	Register Modify — interactively examine/change CPU register contents
T [<Count>]	Trace — execute an instruction, disassemble it, and display the CPU registers
UPLOAD <StartAddress> <EndAddress>	Display memory contents in S-Record format*
VERF [<AddressOffset>]	Verify memory contents against S-Record Data
<RegisterName> <RegisterValue>	Set CPU <RegisterName> to <RegisterValue>
* Refer to Appendix A for S-record information.	

3.5 D-BUG12 COMMAND SET

In the following command descriptions, the examples represent what is seen on the terminal display. For clarity, the user's entry is underlined. This underlining *does not actually appear onscreen*.

A typical example looks like this:

```
>baud 9600                          user's entry
Change Terminal BR, Press Return    D-Bug12's response
>                                   D-Bug12 prompt for next entry
```

ASM Assembler/Disassembler ASM

syntax:

```
ASM  <Address>
```

where:

<Address> is a 16-bit hexadecimal number.

The assembler/disassembler is an interactive memory editor that allows memory contents to be viewed and altered using assembly language mnemonics. Each entered source line is translated into object code and placed into memory at the time of entry. When displaying memory contents, each instruction is disassembled into its source mnemonic form and displayed along with the hexadecimal object code and any instruction operands.

Assembler mnemonics and operands may be entered in any mix of upper and lower case letters. Any number of spaces may appear between the assembler prompt and the instruction mnemonic or between the instruction mnemonic and the operand. Numeric values appearing in the operand field are interpreted as *signed* decimal numbers. Placing a $ in front of any number will cause the number to be interpreted as a hexadecimal number.

When an instruction is disassembled and displayed, the D-Bug12 prompt is displayed following the disassembled instruction. If a carriage return is the first non-space character entered following the prompt, the next instruction in memory is disassembled and displayed on the next line.

If a CPU12 instruction is entered following the prompt, the entered instruction is assembled and placed into memory. The line containing the new entry is erased and the new instruction is disassembled and displayed on the same line. The next instruction location is then disassembled and displayed on the screen.

The instruction mnemonics and operand formats accepted by the assembler follows the syntax as described in the *CPU12 Reference Manual*.

There are a number of M68HC11 instruction mnemonics that appear in the *CPU12 Reference Manual* that do not have directly equivalent CPU12 instructions. These mnemonics, listed in Table 3-2, are translated into functionally equivalent CPU12 instructions. To aid the current M68HC11 users who may desire to continue using the M68HC11 mnemonics, the disassembler portion of the assembler/disassembler recognizes the functionally equivalent CPU12 instructions and disassembles those instructions into the equivalent M68HC11 mnemonics.

When entering branch instructions, the number placed in the operand field should be the absolute destination address of the instruction. The assembler calculates the two’s-complement offset of the branch and places the offset in memory with the instruction

The assembly/disassembly process may be terminated by entering a period (.) as the first non-space character following the assembler prompt.

restrictions:

None.

Table 3-2. M68HC11 to CPU12 Instruction Translation

M68HC11 Mnemonic	CPU12 Instruction	M68HC11 Mnemonic	CPU12 Instruction
CLC	ANCC # $FE	INS	LEAS 1, S
CLI	ANCC # $EF	TAP	TFR A, CC
CLV	ANCC # $FD	TPA	TFR CC, A
SEC	ORCC # $01	TSX	TFR S, X
SEI	ORCC # $10	TSY	TFR S, Y
SEV	ORCC # $02	XGDX	EXG D, X
ABX	LEAX B, X	XGDY	EXG D, Y
ABY	LEAY B, Y	SEX R_8, R_{16}	TFR R_8, R_{16}
DES	LEAS -1, S		

example:

```
>ASM 800

0800  CC1000          LDD       #$1000
0803  1803123401FE    MOVW      #$1234,$01FE
0809  0EF9800001F1    BRSET     -32768,PC,$01,$0700
080F  18FF            TRAP      $FF
0811  183FE3          ETBL      <Illegal Addr Mode>   >.
>
```

a**ssembly operand format:**

This section describes the operand format used by the assembler when assembling CPU12 instructions. The operand format accepted by the assembler is described separately in the *CPU12 Reference Manual*. Rather than describe the numeric format accepted for each instruction, some general rules are used. Exceptions and complicated operand formats are described separately.

In general, anywhere the assembler expects a numeric value in the operand field, either a decimal or hexadecimal value may be entered. Decimal numbers are entered as signed constants having a range of -32768 to 65535. A leading minus sign (-) indicates negative numbers, the absence of a leading minus sign indicates a positive number. A leading plus sign (+) is not allowed.

Hexadecimal numbers must be entered with a leading dollar sign ($) followed by one to four hexadecimal digits. The default number base is decimal.

For all branching instructions (Bcc, LBcc, BRSET, BRCLR, DBEQ, DBNE, IBEQ, IBNE, TBEQ, and TBNE), the number entered as the branch address portion of the operand field is the absolute address of the branch destination. The assembler calculates the two's-complement offset to be placed in the assembled object code.

disassembly operand format:

The operand format used by the disassembler is described separately in the *CPU12 Reference Manual*. Rather than describing the numeric format used for each instruction, some general rules are applied. Exceptions and complicated operand formats are described separately.

All numeric values disassembled as hexadecimal numbers are preceded by a dollar sign ($) to avoid being confused with values disassembled as signed decimal numbers.

For all branch (Bcc, LBcc, BRSET, BRCLR, DBEQ, DBNE, IBEQ, IBNE, TBEQ, TBNE) instructions the numeric value of the address portion of the operand field is displayed as the hexadecimal absolute address of the branch destination.

All offsets used with indexed addressing modes are disassembled as signed decimal numbers.

All addresses, whether direct or extended, are disassembled as four digit hexadecimal numbers.

All 8-bit mask values (BRSET/BRCLR/ANDCC/ORCC) are disassembled as two-digit hexadecimal numbers.

All 8-bit immediate values are disassembled as hexadecimal numbers.

All 16-bit immediate values are disassembled as hexadecimal numbers.

BAUD — Set Baud Rate — BAUD

syntax:

```
BAUD  <BAUDRate>
```

where:

<BAUDRate> is an unsigned 16-bit decimal number.

The BAUD command is used to change the communications rate of the SCI used by D-Bug12 for the terminal interface.

restrictions:

Because the <BAUDRate> parameter supplied on the command line is a 16-bit unsigned integer, BAUD rates greater than 65535 baud cannot be set using this command. The SCI BAUD rate divider value for the requested BAUD rate is calculated using the M clock value supplied in the Customization Data area. Because the SCI BAUD rate divider is a 13-bit counter, certain BAUD rates may not be supported at particular M clock frequencies. If the value calculated for the SCI's BAUD rate divider is equal to zero or greater than 8191, command execution is terminated and the communications BAUD rate is not changed.

example:

```
>BAUD 50

Invalid BAUD Rate
>BAUD 38400
Change Terminal BR, Press Return
>
```

BF Block Fill BF

syntax:

```
BF  <StartAddress>  <EndAddress>  [<Data>]
```

where:

<StartAddress>	is a 16-bit hexadecimal number.
<EndAddress>	is a 16-bit hexadecimal number.
<Data>	is an 8-bit hexadecimal number.

The Block Fill command is used to place a single 8-bit value into a range of memory locations. <StartAddress> is the first memory location written with <data> and <EndAddress> is the last memory location written with <data>. If the <data> parameter is omitted, the memory range is filled with the value $00.

restrictions:

None.

example:

```
>BF 6400 6fff 0
>BF 6f00 6fff 55
>
```

BR Breakpoint Set BR

syntax:

```
BR  [<Address>  <Address>  ...]
```

where:

<Address> are optional 16-bit hexadecimal numbers.

The BR command is used to set a software breakpoint at a specified address or to display any previously set breakpoints. The function of a breakpoint is to halt user program execution when the program reaches the breakpoint address. When a breakpoint address is encountered, D-Bug12 disassembles the instruction at the breakpoint address, prints the CPU12's register contents, and waits for a D-Bug12 command to be entered by the user.

Breakpoints are set by typing the breakpoint command followed by one or more breakpoint addresses. Entering the breakpoint command without any breakpoint addresses will display all the currently set breakpoints.

A maximum of 10 user breakpoints may be set at one time.

restrictions:

D-Bug12 implements the breakpoint function by replacing the instruction opcode at the breakpoint address in the users program with an SWI instruction. For this reason, a breakpoint may not be set on a user SWI instruction. Breakpoints may only be set at an opcode address, and breakpoints may only be placed at memory addresses in modifiable memory.

Even though D-Bug12 supports a maximum of 10 user defined breakpoints, a maximum of 9 breakpoints may be set on the command line at one time. This restriction is due to the limitation of the command line processor, which allows a maximum of 10 command line arguments including the command string.

example:

```
>BR 35ec 2f80 c592
Breakpoints: 35EC 2F80 C592

>BR
Breakpoints: 35EC 2F80 C592

>
```

BULK Bulk Erase On-Chip EEPROM BULK

syntax:

BULK

The BULK command is used to erase the entire contents of the on-chip EEPROM in a single operation. After the bulk erase operation has been performed, each on-chip EEPROM location is checked for an erased condition.

restrictions:

None.

example:

```
>BULK

>
```

CALL — Call Subroutine — CALL

syntax:

```
CALL  [<Address>]
```

where:

<Address> is an optional 16-bit hexadecimal number.

The CALL command is used to execute a subroutine and return to the D-Bug12 monitor program when the final RTS of the subroutine is executed. When control is returned to D–Bug12, the CPU register contents are displayed. All CPU registers contain the values at the time the final RTS instruction was executed, with the exception of the program counter (PC). The PC contains the starting address of the subroutine. If a subroutine address is not supplied on the command line, the current value of the Program Counter (PC) is used as the starting address.

NOTE:

No user breakpoints are placed in memory before execution is transferred to user code.

restrictions:

If the called subroutine modifies the value of the stack pointer during its execution, it MUST restore the stack pointer's original value before executing the final RTS of the called subroutine. This restriction is required because a return address is placed on the user's stack that returns to D-Bug12 when the final RTS of the subroutine is executed. Obviously, any subroutine must obey this restriction to execute properly.

example:

```
>CALL 820
Subroutine Call Returned

 PC    SP    X     Y      D = A:B     CCR = SXHI  NZVC
0820  0A00  057C  0000       0F:F9          1001  0000
>
```

GO — Go Execute a User Program — GO

syntax:

```
G  [<Address>]
```

where:

<Address> is an optional 16-bit hexadecimal number.

The G command is used to begin the execution of user code in real time. Before beginning execution of user code, any breakpoints that were set with the BR command are placed in memory. Execution of the user program continues until a user breakpoint is encountered, a CPU exception occurs, or the EVB's reset or program-abort switch is pressed.

When user code halts for any of these reasons (except reset, which wipes the slate clean) and control is returned to D-Bug12, a message is displayed explaining the reason for user program termination. In addition, D-Bug12 disassembles the instruction at the current PC address, prints the CPU12's register contents, and waits for the next D-Bug12 command to be entered by the user.

If a starting address is not supplied in the command line parameter, program execution will begin at the address defined by the current value of the Program Counter.

restrictions:

None.

example:

```
>G 800
User Breakpoint Encountered

 PC    SP    X     Y      D = A:B   CCR = SXHI  NZVC
0820  09FE  057C  0000      00:00        1001   0100
0820  08            INX
>
```

GT Go Till GT

syntax:

```
GT  <Address>
```

where:

<Address> is a 16-bit hexadecimal number.

The GT command is similar to the G command except that a temporary breakpoint is placed at the address supplied on the command line. Any breakpoints that were set by the use of the BR command are NOT placed in the user code before program execution begins. Program execution begins at the address defined by the current value of the Program Counter. When user code reaches the temporary breakpoint and control is returned to D-Bug12, a message is displayed explaining the reason for user program termination. In addition, D-Bug12 disassembles the instruction at the current PC address, prints the CPU12's register contents, and waits for a command to be entered by the user.

restrictions:

None.

example:

```
>GT 820
Temporary Breakpoint Encountered

 PC    SP    X     Y      D = A:B   CCR = SXHI  NZVC
0820  09FE  057C  0000       00:00        1001  0100
0820  08                     INX
>
```

HELP — Onscreen Help Summary — HELP

syntax:

HELP

The HELP command is used to display a summary of the D-Bug12 command set. Each command is shown along with its command line format and a brief description of its function.

restrictions:

None.

example:

```
>HELP
ASM <Address>  Single line assembler/disassembler
    <CR>       Disassemble next instruction
    < . >      Exit assembly/disassembly
BAUD <baudrate>  Set communications rate for the terminal
BF <StartAddress> <EndAddress> [<data>]  Fill memory with data
BR [<Address>]  Set/Display user breakpoints
BULK  Erase entire on-chip EEPROM contents
CALL [<Address>]  Call user subroutine at <Address>
G [<Address>]  Begin/continue execution of user code
GT <Address>  Set temporary breakpoint at <Address> & execute user code
HELP  Display this D-Bug12 command summary
LOAD [<AddressOffset>]  Load S-Records into memory
MD <StartAddress> [<EndAddress>]  Memory Display Bytes
MDW <StartAddress> [<EndAddress>]  Memory Display Words
MM <StartAddress>  Modify Memory Bytes
   < CR >           Examine/Modify next location
   < / > or < = >   Examine/Modify same location
   < ^ > or < – >    Examine/Modify previous location
   < . >            Exit Modify Memory command
MMW <StartAddress>  Modify Memory Words (same subcommands as MM)
MOVE <StartAddress> <EndAddress> <DestAddress>  Move a block of memory
NOBR [<address>]  Remove One/All Breakpoint(s)
RD  Display all CPU registers
RM  Modify CPU Register Contents
T [<count>]  Trace <count>  Instructions
UPLOAD <StartAddress> <EndAddress>  S-Record Memory display
VERF [<AddressOffset>]  Verify S-Records against memory contents
<Register Name> <Register Value>  Set register contents
   Register Names: PC, SP, X, Y, A, B, D
   CCR Status Bits: S, XM, H, IM, N, Z, V, C
>
```

LOAD — Load S-Record File — LOAD

syntax:

```
LOAD  [<AddressOffset>]
{Send File}
```

where:

<AddressOffset>	is an optional 16-bit hexadecimal number.
{*Send File*}	is the host-computer communications program's utility for sending an ASCII (text) file. Refer to Appendix B for examples.

The Load command is used to load S-Record object files into memory from an external device. The address offset, if supplied, is added to the load address of each S-Record before its data bytes are placed in memory. Providing an address offset other than zero allows object code or data to be loaded into memory at a location other than that for which it was assembled. During the loading process, the S-Record data is not echoed to the control console. However, for each ten S-Records that are successfully loaded, an ASCII asterisk character (*) is sent to the control console, When an S-Record file has been successfully loaded, control returns to the D-Bug12 prompt.

The Load command is terminated when D-Bug12 receives an 'S9' end of file record. If the object file being loaded does not contain an 'S9' record, D–Bug12 does not return its prompt and continues to wait for the end of file record. Pressing the Reset switch returns D–Bug12 to its command line prompt.

restrictions:

None.

example:

```
>LOAD 1000
**********
>
```

MD — Memory Display — MD

syntax:

MD <StartAddress> [<EndAddress>]

where:

<StartAddress>	is a 16-bit hexadecimal number.
<EndAddress>	is an optional 16-bit hexadecimal number.

The Memory Display command displays the contents of memory as both hexadecimal bytes and ASCII characters, 16-bytes on each line. The <StartAddress> parameter must be supplied; the <EndAddress> parameter is optional. When the <EndAddress> parameter is not supplied, a single line is displayed.

The number supplied as the <StartAddress> parameter is rounded down to the next lower multiple of 16, while the number supplied as the <EndAddress> parameter is rounded up to the next higher multiple of 16 - 1. This causes each line to display memory in the range of $xxx0 through $xxxF. For example, if $205 is entered as the start address and $217 as the ending address, the actual memory range displayed would be $200 through $21F.

restrictions:

None.

example:

```
>MD 800
0800  AA 04 37 6A - 00 06 27 F9 - 35 AE 78 0D - B7 56 78 20  ..7j..'.5.x..Vx

>MD 800 87f
0800  AA 04 37 6A - 00 06 27 F9 - 35 AE 78 0D - B7 56 78 20  ..7j..'.5.x..Vx
0810  B6 36 27 F9 - 35 AE 27 F9 - 35 9E 27 F9 - 35 BE B5 28  .6'.5.'.5.'.5..(
0820  27 F9 35 D6 - 37 B8 00 0F - 37 82 01 0A - 37 36 FF F0  '.5.7...7...76..
0830  7C 10 37 B3 - 00 00 37 B6 - 00 0F AA 04 - A5 02 37 B6  |.7...7.......7.
0840  00 0F 27 78 - 37 6A 00 06 - 27 F9 35 78 - 27 F9 35 56  ..'x7j..'.5x'.5V
0850  78 0D B7 10 - 78 3B 37 86 - 00 DC 27 F9 - 35 48 78 57  x...x;7...'.5HxW
0860  37 86 00 DE - F5 01 EA 09 - 37 B5 0D 0A - 27 F9 36 2A  7.......7...'.6*
0870  A5 00 37 65 - 00 02 27 F9 - 35 E8 37 9C - 37 4C F5 02  ..7e..'.5.7.7L..
>
```

MDW Memory Display, Word MDW

syntax:

```
MDW  <StartAddress>  [<EndAddress>]
```

where:

<StartAddress>	is a 16-bit hexadecimal number.
<EndAddress>	is an optional 16-bit hexadecimal number.

The Memory Display Word command displays the contents of memory as hexadecimal words and ASCII characters, 16-bytes on each line. The <StartAddress> parameter must be supplied; the <EndAddress> parameter is optional. When the <EndAddress> parameter is not supplied, a single line is displayed.

The number supplied as the <StartAddress> parameter is rounded down to the next lower multiple of 16, while the number supplied as the <EndAddress> parameter is rounded up to the next higher multiple of 16 - 1. This causes each line to display memory in the range of $xxx0 through $xxxF. For example, if $205 is entered as the start address and $217 as the ending address, the actual memory range displayed would be $200 through $21F.

restrictions:

None.

example:

```
>MDW 800
0800  AA04 376A - 0006 27F9 - 35AE 780D - B756 7820  ..7j..'.5.x..Vx

>MDW 800 87f
0800  AA04 376A - 0006 27F9 - 35AE 780D - B756 7820  ..7j..'.5.x..Vx
0810  B636 27F9 - 35AE 27F9 - 359E 27F9 - 35BE B528  .6'.5.'.5.'.5..(
0820  27F9 35D6 - 37B8 000F - 3782 010A - 3736 FFF0  '.5.7...7...76..
0830  7C10 37B3 - 0000 37B6 - 000F AA04 - A502 37B6  |.7...7.......7.
0840  000F 2778 - 376A 0006 - 27F9 3578 - 27F9 3556  ..'x7j..'.5x'.5V
0850  780D B710 - 783B 3786 - 00DC 27F9 - 3548 7857  x...x;7...'.5HxW
0860  3786 00DE - F501 EA09 - 37B5 0D0A - 27F9 362A  7.......7...'.6*
0870  A500 3765 - 0002 27F9 - 35E8 379C - 374C F502  ..7e..'.5.7.7L..
>
```

MM Memory Modify MM

syntax:

```
MM  <Address>  [<Data>]
```

where:

<Address>	is a16-bit hexadecimal number.
<Data>	is an optional 8-bit hexadecimal number.

The Memory Modify command allows the contents of memory to be examined and/or modified as 8-bit hexadecimal data. If the 8-bit data parameter is present on the command line, the byte at memory location <Address> is replaced with <Data> and the command is terminated. If not, D-Bug12 enters the interactive memory modify mode. In the interactive mode, each byte is displayed on a separate line following the data's address. Once the memory modify command has been entered, single-character sub-commands are used for the modification and verification of memory contents. These sub-commands have the following format:

[<Data>] <CR>	Optionally update current location and display the next location.
[<Data>] </> or <=>	Optionally update current location and redisplay the current location.
[<Data>] <^> or <->	Optionally update current location and display the previous location.
[<Data>] <.>	Optionally update current location and exit Memory Modify.

With the exception of the carriage return, the sub-command must be separated from any entered data with at least one space character. If an invalid sub-command character is entered, an appropriate error message is issued and the contents of the current memory location are redisplayed.

restrictions:

None.

example:

```
>MM 800
0800  00  <CR>
0801  F0  FF
0802  00  ^
0801  FF  <CR>
0802  00  <CR>
0803  08  55 /
0803  55  .
>
```

MMW Memory Modify, Word MMW

syntax:

```
MMW  <Address>  [<Data>]
```

where:

<Address>	is a 16-bit hexadecimal number.
<Data>	is an optional 16-bit hexadecimal number.

The Memory Modify Word command allows the contents of memory to be examined and/or modified as 16-bit hexadecimal data. If the 16-bit data parameter is present on the command line, the word at memory location <Address> is replaced with <Data> and the command is terminated. If not, D-Bug12 enters the interactive memory modify mode. In the interactive mode, each word is displayed on a separate line following the data's address. Once the memory modify command has been entered, single-character sub-commands are used for the modification and verification of memory contents. These sub-commands have the following format:

[<Data>] <CR>	Optionally update current location and display the next location.
[<Data>] </> or <=>	Optionally update current location and redisplay the current location.
[<Data>] <^> or <->	Optionally update current location and display the previous location.
[<Data>] <.>	Optionally update current location and exit Memory Modify.

With the exception of the carriage return, the sub-command must be separated from any entered data with at least one space character. If an invalid sub-command character is entered, an appropriate error message is issued and the contents of the current memory location are redisplayed.

restrictions:

None.

example:

```
>MMW 800
0800  00F0  <CR>
0802  0008  AA55 /
0804  843F  ^
0802  AA55  <CR>
0804  843F  <CR>
0806  C000  .
>
```

MOVE — Move Memory Block — MOVE

syntax:

```
MOVE  <StartAddress>  <EndAddress>  <DestAddress>
```

where:

<StartAddress>	is a 16-bit hexadecimal number.
<EndAddress>	is a 16-bit hexadecimal number.
<DestAddress>	is a 16-bit hexadecimal number.

The MOVE command is used to move a block of memory from one location to another, one byte at a time. The number of bytes moved is one more than the <EndAddress> - <StartAddress>. The block of memory beginning at the destination address may overlap the memory block defined by the <StartAddress> and <EndAddress>.

One of the uses of the MOVE command might be to copy a program from RAM into the on-chip EEPROM memory.

restrictions:

A minimum of one byte may be moved if the <StartAddress> is equal to the <EndAddress>. The maximum number of bytes that may be moved is 2^{16} - 1.

example:

```
>MOVE 800 8ff 1000
>
```

NOBR Remove Breakpoints NOBR

syntax:

```
NOBR  [<Address>  <Address>  ...]
```

where:

<Address> is an optional 16-bit hexadecimal number.

The NOBR command can be used to remove one or more previously entered breakpoints. If the NOBR command is entered without any arguments, all user breakpoints are removed from the breakpoint table.

restrictions:

None.

example:

```
>BR 800 810 820 830
Breakpoints: 0800  0810  0820  0830

>NOBR 810 820
Breakpoints: 0800  0830

>NOBR
All Breakpoints Removed

>
```

RD — Register Display — RD

syntax:

```
RD
```

The Register Display command is used to display the CPU12's registers.

restrictions:

None.

example:

```
>RD
 PC    SP     X    Y       D = A:B   CCR = SXHI NZVC
0206  03FF  1000  3700        27:FF        1001 0001
>
```

RM Register Modify RM

syntax:

```
RM
```

The Register Modify command is used to examine and/or modify the contents of the CPU12's registers in an interactive manner. As each register and its contents is displayed, D-Bug12 allows the user to enter a new value for the register in hexadecimal. If modification of the displayed register is not desired, entering a carriage return will cause the next CPU12 register and its contents to be displayed on the next line. When the last of the CPU12's registers has been examined and/or modified, the RM command displays the first register, giving the user an opportunity to make additional modifications to the CPU12's register contents. Typing a period (.) as the first non space character on the line will exit the interactive mode of the register modify command and return to the D-Bug12 prompt. The registers are displayed in the following order, one register per line: PC, SP, X, Y, A, B, CCR.

restrictions:

None.

example:

```
>RM
PC=0206 200
SP=03FF <CR>
X=1000 1004
Y=3700 <CR>
A=27 <CR>
B=FF <CR>
CCR=D0 D1
PC=0200 .
>
```

T Trace T

syntax:

```
T  [<Count>]
```

where:

<Count> is an optional 8-bit decimal number in the range 1 to 255.

The Trace command is used to execute one or more user program instructions beginning at the current Program Counter (PC) location. As each program instruction is executed, the CPU12's register contents are displayed and the *next* instruction to be executed is displayed. A single instruction may be executed by entering the trace command immediately followed by a carriage return.

restrictions:

Because of the method used to execute a single instruction, branch instructions (Bcc, LBcc, BRSET, BRCLR, DBEQ/NE, IBEQ/NE, TBEQ/NE) that contain an offset that branches back to the instruction opcode DO NOT execute. The terminal appears to become stuck at the branch instruction and does not execute the instruction even if the condition for the branch instruction is satisfied. This limitation can be overcome by using the GT (Go Till) command to set a temporary breakpoint at the instruction following the branch instruction.

When the CPU12 is not operating in background debug mode, there is no specialized hardware available to execute a single instruction. The Trace command makes use of temporary software breakpoints as a means to control CPU execution. For this reason, only instructions that reside in alterable memory may be executed with the Trace command.

example:

```
>T

 PC    SP    X     Y        D=A:B  CCR=SXHI NZVC
0803  09FE  057C  0000      10:00      1001 0000
0803  830001        SUBD    #$0001
>T 3

 PC    SP    X     Y        D=A:B  CCR=SXHI NZVC
0806  09FE  057C  0000      0F:FF      1001 0000
0806  26FB          BNE     $0803

 PC    SP    X     Y        D=A:B  CCR=SXHI NZVC
0803  09FE  057C  0000      0F:FF      1001 0000
0803 830001         SUBD    #$0001
```

```
 PC    SP    X     Y        D=A:B  CCR=SXHI NZVC
0806  09FE  057C  0000      0F:FE      1001 0000
0806  26FB           BNE     $0803
>
```

UPLOAD Display Memory in S-Record Format UPLOAD

syntax:

```
UPLOAD  <StartAddress>  <EndAddress>
```

where:

<StartAddress>	is a 16-bit hexadecimal number.
<EndAddress>	is a 16-bit hexadecimal number.

The UPLOAD command is used to display the contents of memory in Motorola S-Record format. In addition to displaying the specified range of memory, the UPLOAD command also outputs an S9 end-of-file record. The output of this command may be captured by the users terminal program and saved to a disk file.

restrictions:

None.

example:

```
>upload 400 5ff
S123040000F0000843FC0000F50F379F37BF43FCF50F27FA757F177AFA047504177AFA21C5
S123042037B500FF37FAFB0437B5400037FAFB061735FB0037B500C137FAFA003715379C01
S1230440F50F379D37BC012C37BD400085009A003C023D02377C0140B6EE7A0F400037B583
S1230460000337FAFA4C37FAFA5037FAFA5437B5502037FAFA4E37B5302037FAFA5237B58A
S1230480682037FAFA5637BD014037BC000095008A003C023D02377D0172B6EE37BD017259
S12304A037BC020095008A003C023D02377D018EB6EE27F937B0F50F379C37BC00CE27F901
S12304C000FC27F9104C27F90E68378000BE0A0D442D4275673136205631 2E3033202D20E3
S12304E04465627567204D6F6E69746F7220466F7220546865204D36384843316204661ED
S12305006D696C790A0D2843292031393932204D6F746F726F6C612053656D69636F6E64BD
S1230520756374 6F7220496E632E000037B5FF0237FAFA4837B578B037FAFA4A7A0F005E52
S123054000000000000000000200020402080 20C0210000000000000000000000002144F
S123056000000000000000000000000000000000002187A0F3BAC7A0F3BBC7A0F11E87A0F62
S12305803C727A0F3C847A0F3C967A0F3CA8F50F379C379D379E27FAF50F379F37BF43FCE8
S12305A07501177A4054173540523604361C27F90088B0D637BC01BC360227F70A0D3E00A9
S12305C04500B70427F936BC3C01B0F027F7277537BC400017BC405027F936CC780DB60477
S12305E027F936A0274A27F77803B6FEB03A7808B6162776B7DE3730000127F93686752002
S9030000FC
>
```

VERF Verify S-Record File against Memory VERF

syntax:

```
VERF  [<AddressOffset>]
{Send File}
```

where:

<AddressOffset>	is an optional 16-bit hexadecimal number.
{*Send File*}	is the host-computer communications program's utility for sending an ASCII (text) file. Refer to Appendix B for examples.

The VERF command is used to compare the data contained in an S-Record object file to the contents of EVB memory. The address offset, if supplied, is added to the load address of each S-Record before an S-Record's data bytes are compared to the contents of memory. Providing an address offset other than zero allows the S-Record's object code or data to be compared against memory other than that for which the S-Record was assembled.

During the verification process, an ASCII asterisk character (*) is sent to the control console for each ten S-Records that are successfully verified. When an S-Record file has been successfully verified, control returns to the D-Bug12 prompt.

If the contents of EVB memory do not match the corresponding data in the received S-Records, an error message is displayed and the Verify command is terminated. D-Bug12 then returns to its command-line prompt. If the host computer continues to send S-Records to the EVB, D-Bug12 tries to interpret each S-Record as a command and issues error message for each S-Record received.

If the contents of EVB memory match the contents of the received S-Records, the Verify command terminates when D-Bug12 receives an S9 end-of-file record. If the object file being verified does not contain an S9 record, D-Bug12 continues to wait for an S9 record without returning to the command-line prompt. Pressing the reset switch, S1, returns D-Bug12 to its command-line prompt.

restrictions: None.

example:

```
>VERF 1000
********************
>
```

<Register Name> Modify Register Value <Register Name>

syntax:

```
<RegisterName>  <RegisterValue>
```

where:

<RegisterName> is one of the CPU12 registers listed in Table 3-3.

<RegisterValue> is an 8- or 16-bit hexadecimal number.

Table 3-3. CPU12 Registers

Register Name	Description	Legal Range
PC	Program Counter	$0 to $FFFF
SP	Stack Pointer	$0 to $FFFF
X	X-Index Register	$0 to $FFFF
Y	Y-Index Register	$0 to $FFFF
A	A Accumulator	$0 to $FF
B	B Accumulator	$0 to $FF
D	D Accumulator (A:B)	$0 to $FFFF
CCR	Condition Code Register	$0 to $FF

Each of the fields in the Condition Code Register (CCR) may be modified by using the bit names in Table 3-4.

Table 3-4. Condition Code Register Bits

CCR Bit Name	Description	Legal Values
S	STOP Enable	0 or 1
H	Half Carry	0 or 1
N	Negative Flag	0 or 1
Z	Zero Flag	0 or 1
V	Two's Complement Overflow Flag	0 or 1
C	Carry Flag	0 or 1
IM	IRQ Interrupt Mask	0 or 1
XM	XIRQ Interrupt Mask	0 or 1

This set of "commands" uses a CPU12 register name as the command name to allow changing the register's contents. Each register name or CCR bit name is entered on the command line followed by a space, then followed by the new register or bit contents. After successful alteration of a CPU register or CCR bit, the entire CPU register set is displayed.

restrictions:

None.

example:

```
>PC 700e

 PC    SP    X     Y        D=A:B  CCR=SXHI  NZVC
700E  0A00  7315  7D62       47:44    1001   0000
>X 1000

 PC    SP    X     Y        D=A:B  CCR=SXHI  NZVC
700E  0A00  1000  7D62       47:44    1001   0000
>C 1

 PC    SP    X     Y        D=A:B  CCR=SXHI  NZVC
700E  0A00  1000  7D62       47:44     1001   0001
>Z 1

 PC    SP    X     Y        D=A:B  CCR=SXHI  NZVC
700E  0A00  1000  7D62       47:44    1001   0101
>D adf7

 PC    SP    X     Y        D=A:B  CCR=SXHI  NZVC
700E  0A00  1000  7D62       AD:F7    1001   0101
>
```

3.6 ALTERNATE EXECUTION FROM EEPROM

In this hardware-configured mode (pins 1-2 jumpered on header W20), the EVB begins operation out of reset by executing the user program in on-chip EEPROM starting at address $1000, as shown in Table 3-5.

This mode is effected using the MCU's PAD0 line, which is broken out in J9 for possible custom use in the prototype area.

Control can be returned to D-Bug12 in the following ways:

1. Move the jumper on W20 to pins 2-3 and reset the EVB. *Do not activate the program abort function — see note in section 3.3.*
2. Terminate the user program with code that returns to D-Bug12 after execution has finished.

To return to D-Bug12 after a user program has finished, include the following lines as the last instructions to be executed in the program:

```
STACKTOP:   equ    $0c00       ; stack at top of on-chip RAM
DEBUG12:    equ    $FD90
;
            lds    #STACKTOP
            jmp    DEBUG12     ; jump to start of D-Bug12 code
```

3.7 OFF-BOARD CODE GENERATION

To generate a user program on a host computer and load it into the EVB's memory, follow these steps:

NOTE

For steps 2 and 3, follow the instructions in the *Motorola Microcontroller Families MCUasm User's Manual.*

1. Set up the EVB system with a host computer as the terminal — see section 2.5.3.
2. In the host computer's native operating mode — i.e., *before* starting the communications program that allows it to serve as the EVB's terminal — write and assemble the program using Motorola's MCUasm assembler.
3. Using the MCUasm assembler's HEX utility, generate a Motorola S-Record file from the object (.HEX) file. Appendix A contains detailed information about the S-Record formats.
4. Start the EVB with D-Bug12 as the default operating mode, using the procedure in section 3.1.

5. At the D-Bug12 prompt, issue D-Bug12's LOAD command with any parameters. Note that this requires interaction with the terminal communications program's "send file" utility — see Appendix B for examples.

3.8 MEMORY USAGE

3.8.1 Description

The EVB's memory usage and requirements are described below and summarized in Table 3-5. Note that this memory mapping applies only to the factory-default memory configuration.

The monitor program, D-Bug12, occupies 24 Kbytes in the two 32 Kbyte EPROMs, U7 and U9A. The remaining 8 Kbytes are available for user programs and utilities, but since this ROM area cannot be directly written, special techniques are required to take advantage of it. For information on using it, refer to **Appendix E Customizing the EPROMs.**

Since the MCU must manage the execution of D-Bug12 and other EVB functions, 512 bytes of on-chip RAM, from $0A00 to $0BFF, are required for stack and variable storage. The remaining 512 bytes of on-chip RAM, from $0800 to $09FF, are available for variable storage and stack space by user programs.

NOTE

D-Bug12 sets the default value of the user's stack pointer to $0A00. This is not a mistake. The M68HC12's stack pointer points to the last byte that was pushed onto the stack, rather than to the next available byte on the stack, as the M68HC11 does. The M68HC12 first decrements its stack pointer, then stores data on the stack. The M68HC11 stores data on the stack and then decrements its stack pointer.

The 16 Kbytes of external RAM, from $4000 to $7FFF, are available for user code and data.

3.8.2 Memory Map

Table 3-5. Factory-Configuration Memory Map

Address Range	Description	Location
$0000 - $01FF	CPU registers	on-chip (MCU)
$0800 - $09FF $0A00 - $0BFF	user code/data reserved for D-Bug12	1K on-chip RAM (MCU)
$1000 - $1FFF	user code/data	4K on-chip EEPROM (MCU)
$4000 - $7FFF	user code/data	16K external RAM (U4, U6A)
$8000 - $9FFF $A000 - $FD7F $FD80 - $FDFF $FE00 - $FE7F $FE80 - $FEFF $FF00 - $FF7F $FF80 - $FFFF	available for user programs* D-Bug12 program D-Bug12 startup code* user-accessible functions D-Bug12 customization data* available for user programs* reserved for interrupt and reset vectors	32K external EPROM (U7, U9A)
*Code in these areas may be modified. Requires reprogramming of the EPROMs — refer to **Appendix E Customizing the EPROMs**.		

3.9 OPERATIONAL LIMITATIONS

D-Bug12 and other EVB functions require some of the MC68HC812A4's resources for management. For this reason, the EVB cannot provide true emulation of a target system. These limitations are described in the following sections.

3.9.1 On-Chip RAM

D-Bug12 requires 512 bytes of on-chip RAM for stack and variable storage. This usage is shown in Table 3-5.

3.9.2 SCI Port Usage

D-Bug12 requires one of the MCU's Serial Communications Interface (SCI) ports for the terminal interface. The SCI port used for this purpose is jumper-selectable (W14), but the one selected is unavailable for other uses.

3.9.3 Dedicated MCU Pins

As used on the EVB with D-Bug12, the following MCU lines perform specific functions. If an application requires their use, the EVB hardware and/or operating software must be custom-configured, or special precautions must be taken in the application code to avoid conflicts with the D-Bug12 usage.

PE0/XIRQ* — program-abort function (S2). Additionally, there are two software limitations on the program-abort function:

1. D-Bug12 enables the hardware XIRQ* interrupt by initializing the XM bit in the Condition Code Register (see Table 3-4). If this interrupt is subsequently disabled in software, for example with the D-Bug12 RM command, it cannot be directly enabled again.
2. If the user code replaces the D-Bug12 interrupt handler with one of its own, the program-abort function is effectively disabled.

PAD0 — selects normal or alternate execution mode (W20)

PAD1 — selects the SCI port used for the terminal interface (W14).

PF4/CSD* and PF5/CSP0* — dedicated to chip-select usage. Not available for I/O in the default configuration.

Ports A, B, C, D, and G — dedicated to address/data bus usage. Not available as I/O ports in the default configuration.

3.9.4 Terminal Communications

High baud rates occasionally result in dropped characters on the terminal display. This is not the result of a baud rate mismatch; it is due to the host processor being too busy or too slow to process incoming data at the selected baud rate. The D-Bug12 MD, MDW, T, and HELP commands may be affected by this problem. Sometimes the problem can be ignored without harm. If it requires correcting, try the following:

- Use a slower baud rate.
- Try a different communications program.

- In multitasking environments such as Windows 3.1 and the MacIntosh System 7, the problem can occur when several applications are running at once. Try closing unnecessary applications or exiting Windows.
- When using the MD, MDW, or T commands, try displaying fewer address locations or tracing fewer instructions at a time.

CHAPTER 4
HARDWARE REFERENCE

4.1 PCB DESCRIPTION

The EVB printed circuit board (PCB) is an 8-inch by 8-inch board with six layers — one power, one ground, and four signal layers. The signal layers containing **cut-trace header footprints**, described in section 4.2, comprise the top and bottom layers for accessibility.

Most of the connection points on the EVB are headers on 1/10-inch centers, with the following exceptions:

- Subminiature D connectors for the SCI RS-232C interfaces
- Loop-style hardware connections for test points
- External power-supply connections

4.2 CONFIGURATION HEADERS AND JUMPER SETTINGS

The EVB is designed for maximum flexibility — there are 45 PCB footprints available for configuration headers. These are of two types:

Factory-installed headers are those most likely to be used for configuration without major alteration of the EVB's hardware operation. These headers are populated, and the factory-installed jumpers on them are preset for the default EVB hardware and firmware (D-Bug12) configurations. Table 4-1 lists these headers by function and describes their default and optional jumper settings.

Cut-trace header footprints offer EVB hardware options that are less likely to be changed. These footprints are not populated. The default connection between pins is a trace on the PCB. To change a cut-trace footprint, the PCB trace must be cut. To return to the original configuration, a header and a jumper must be installed to re-establish the shunt

NOTE

Use of the cut-trace header footprints requires a thorough understanding of the MCU and of the EVB hardware. Refer to the *MC68HC812A4 Technical Summary* and to the EVB schematic diagram for design information.

CAUTION

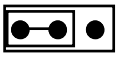
When cutting a PCB trace to customize a header footprint, be careful not to cut adjacent traces. Do not damage the underlying PCB layers by cutting too deeply.

Key to Table 4-1:

2-pin header with no jumper installed

2-pin header with jumper installed

3-pin header with no jumper installed



3-pin header with jumper installed on left 2 pins

1-2 **bold** pin numbers indicate factory-default settings

Table 4-1. Jumper-Selectable Functions

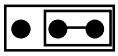
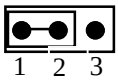
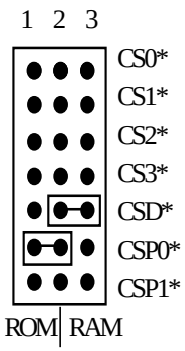
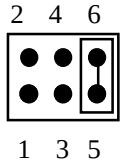
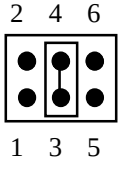
Diagram	Setting	Description
W1 Low-Voltage Inhibit (LVI)		
	1-2 off	low-voltage inhibit is enabled low-voltage inhibit is disabled
W3 RAM Write-Protection		
	**1-2** 2-3	RAM write-protection is disabled RAM write-protection is enabled
W10 TXD1 — RS-232C Transmit Data (TXD) Enable, SCI Port 1		
	**1-2** 2-3	TXD on SCI port 1 is enabled TXD on SCI port 1 is disabled
W11 ROM and RAM Chip Select (CS)		
	1-2 2-3	connects an MCU chip select to the devices installed in the ROM sockets connects an MCU chip select to the devices installed in the RAM sockets DEFAULT: CSP0* is the ROM chip select CSD* is the RAM chip select
W12(1) RAM Pin Assignment — pin 30 of 32-pin package *or* pin 28 of 28-pin package		
	1-2 3-4 **5-6**	pin is connected to MCU address line A17 — for Narrow modes pin is connected to MCU address line A18 — for Wide modes pin is connected to V_{DD} — for 28-pin devices
W13(1) RAM Pin Assignment — pin 28 of 32-pin package *or* pin 26 of 28-pin package		
	1-2 **3-4** 5-6	pin is connected to MCU address line A13 — for Narrow modes pin is connected to MCU address line A14 — for Wide modes pin is connected to V_{DD} — for the device's chip enable (CE2)

Table 4-1. Jumper-Selectable Functions (continued)

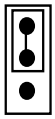
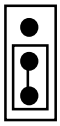
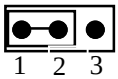
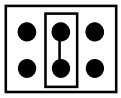
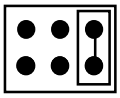
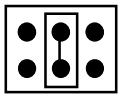
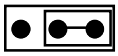
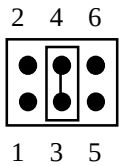
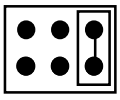
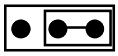
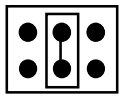
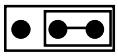
Diagram	Setting	Description
W14 SCI Port Assignment to Terminal Interface		
1 2 3 	**1-2** 2-3	SCI port 0 serves as the D-Bug12 terminal interface SCI port 1 serves as the D-Bug12 terminal interface
W20 D-Bug12 (normal) or EEPROM (alternate) Execution Mode		
1 2 3 	1-2 **2-3**	the code in on-chip EEPROM is executed out of reset D-Bug12 is executed out of reset
W21 TXD0 — RS-232C Transmit Data (TXD) Enable, SCI Port 0		
 1 2 3	**1-2** 2-3	TXD on SCI port 0 is enabled TXD on SCI port 0 is disabled
W22(2) ROM Pin Assignment — pin 31 of 32-pin package		
2 4 6  1 3 5	1-2 **3-4** 5-6	pin is connected to MCU address line A18 — for Narrow modes pin is connected to MCU address line A19 — for Wide modes pin is connected to V_{DD} — to disable the device's write enable (WE*)
W24(2) ROM Pin Assignment —pin 30 of 32-pin package *or* pin 28 of 28-pin package		
2 4 6  1 3 5	1-2 3-4 **5-6**	pin is connected to MCU address line A17 — for Narrow modes pin is connected to MCU address line A18 — for Wide modes pin is connected to V_{DD} — for 28-pin devices
W29(2) ROM Pin Assignment —pin 29 of 32-pin package *or* pin 27 of 28-pin package		
2 4 6  1 3 5	1-2 **3-4** 5-6	pin is connected to MCU address line A14 — for Narrow modes pin is connected to MCU address line A15 — for Wide modes pin is connected to V_{DD} — to disable the device's write enable (WE*)
W30(3) MCU Background Mode Select		
 1 2 3	1-2 **2-3**	MCU's BKGD pin is connected to V_{SS} MCU's BKGD pin is connected to V_{DD}

Table 4-1. Jumper-Selectable Functions (continued)

Diagram	Setting	Description
W32(2) ROM Pin Assignment —pin 28 of 32-pin package *or* pin 26 of 28-pin package		
	1-2 **3-4** 5-6	pin is connected to MCU address line A13 — for Narrow modes pin is connected to MCU address line A14 — for Wide modes pin is connected to V_{DD} — to enable the device's chip enable (CE2)
W33(2) ROM Pin Assignment —pin 3 of 32-pin package *or* pin 1 of 28-pin package		
	1-2 3-4 **5-6**	pin is connected to MCU address line A15 — for Narrow modes pin is connected to MCU address line A16 — for Wide modes pin is connected to V_{DD} — for ROM program voltage (V_{PP})
W34(3) MCU MODB Select		
	1-2 **2-3**	MCU's PE6/MODB pin is connected to V_{SS} MCU's PE6/MODB pin is connected to V_{DD}
W36(2) ROM Pin Assignment — pin 2 of 32-pin package		
	1-2 **3-4** 5-6	pin is connected to MCU address line A16 — for Narrow modes pin is connected to MCU address line A17 — for Wide modes pin is connected to V_{DD}
W42(3) MCU MODA Select		
	1-2 **2-3**	MCU's PE5/MODA pin is connected to V_{SS} MCU's PE5/MODA pin is connected to V_{DD}
NOTES:		
(1) W12 and W13 together select the type of RAM installed. (2) W22, W24, W29, W32, W33, and W36 together select the type of ROM installed. (3) W30, W34, and W42 together determine the MCU's mode of operation.		

4.3 POWER INPUT CIRCUITRY

The input power connector on the EVB is a 2-pin, lever-actuated connector (J6), illustrated in Figure 2-1. Fuse F1 (1.5 amp), Zener diode VR1, and diode CR1 provide over-voltage and reverse-polarity protection. Decoupling capacitors filter ripple and noise from the supply voltage. A red LED (DS1) serves as the power-on indicator.

Cut-trace header footprints (see section 4.2) on the EVB allow isolating the V_{SS} (ground) and V_{DD} (+Vdc) power circuits for different functional areas. These individually filtered circuits can then be connected to separate power sources. This can be helpful for purposes such as power-usage analysis. The following power circuits can be isolated:

- V_{SSI} / V_{DDI} — MCU core usage
- V_{SSEX0} / $V_{DDEX0,}$ V_{SSEX1} / $V_{DDEX1,}$ V_{SSEX2} / V_{DDEX2} — three separate circuits for MCU I/O pins
- V_{SSPLL} / V_{DDPLL} — Phase-Locked Loop (PLL)
- V_{SSA} / $V_{DDA,}$ V_{RL} / V_{RH} — A/D Converter power and reference voltages

Refer to the EVB schematic diagram to locate the cut-trace header footprint that isolate these circuits.

4.4 TERMINAL INTERFACE

An RS-232C transceiver (U5B) links the MCU's two Serial Communications Interfaces (SCI0 and SCI1) with separate RS-232C ports on the EVB. One of these ports (SCI0 by default) serves as the terminal interface for D-Bug12 operation. The other port is available for user applications. The communications parameters for these ports are described in **2.5 Terminal Communications Setup**.

.There are two possible connectors for each port — a right-angle DB-9 receptacle wired as DCE (for standard RS-232C cabling) and a functionally equivalent 3-pin header (for customized cabling). SCI0 uses connectors J3 or J4; SCI1 uses connectors J1 or J2. The pin assignments for these connectors are listed in Table 2-1. Note that the EVB's serial ports use only three of the RS-232C signals: Receive Data (RXD), Transmit Data (TXD), and Ground (GND).

To change the D-Bug12 terminal port from SCI0 (the factory default) to SCI1, move the jumper on header W14 to pins 2-3, as shown in Table 4-1. Header J1 can then be used for the terminal port connection without further hardware modification. If a standard RS-232C cable connection is needed for this port, install a right-angle DB-9 receptacle in the footprint for J2 (not populated at the factory).

The EVB's RS-232C output signals (Transmit Data) can be disabled by setting the jumpers on headers W10 and W21, as shown in Table 4-1.

4.5 MICROCONTROLLER

The MC68HC812A4 is the first of a family of next generation M68HC11 microcontrollers with on-chip memory and peripheral functions. The CPU12 is a high-speed, 16-bit processing unit. The programming model and stack frame are identical to those of the standard M68HC11 CPU. The CPU12 instruction set is a proper superset of the M68HC11 instruction set. All M68HC11 instruction mnemonics are accepted by CPU12 assemblers with no changes.

The EVB-resident MC68HC812A4 (U8) has seven modes of operation. These modes are determined at reset by the state of three mode pins — BKGD, MODB, and MODA — as shown in Table 4-2.

The EVB is factory-configured for MCU operation in the Normal Expanded Wide (x16) mode. In this mode of operation, the expanded bus is present with a 16-bit data bus. Port D is the low byte data bus and Port C is the high byte data bus. Table 3-5, the Factory-Configuration Memory Map, lists the MCU resource usage in this default configuration.

In the Normal Expanded Narrow (x8) mode of operation, the expanded bus is present with an 8-bit data bus. Port C functions as the data bus in this mode. Port D is available for general purpose I/O.

In the Normal Single Chip mode of operation, no external bus is available. All program and data fetches are from on-chip memory or peripheral registers. Ports A, B, C, and D are available for general purpose I/O.

The Special Peripheral mode of operation is a test mode. The CPU is not active. On-chip peripherals may be accessed directly by an external bus master. It is not possible to change from or to this mode without going through reset.

The Special Expanded Wide, Special Expanded Narrow, and Special Single Chip modes provide basically the same functionality as the respective normal modes. These special modes are primarily for testing and provide access to several key features, including:

Special Expanded Narrow — to view 16-bit accesses without changing the instruction cycle times, port D may be used to view the upper 8 bits of the data bus.

Special Single Chip — background debug mode is immediately active out of reset. Execution begins from the background debug ROM. Commands are sent to the CPU through the background debug interface pin. A background debug interface is required, as described in section 4.12.

For more information on the CPU, refer to the *CPU12 Reference Manual*.

Table 4-2. CPU Mode Selection

BKGD Header W30	MODB Header W34	MODA Header W42	Mode Description
0 (2)	0 (2)	0 (2)	Special Single Chip
0 (2)	0 (2)	1 (1)	Special Expanded Narrow
0 (2)	1 (1)	0 (2)	Special Peripheral
0 (2)	1 (1)	1 (1)	Special Expanded Wide
1 (1)	0 (2)	0 (2)	Normal Single Chip
1 (1)	0 (2)	1 (1)	Normal Expanded Narrow
1 (1)	1 (1)	0 (2)	Reserved (currently defaults to peripheral mode)
1 (1)	1 (1)	1 (1)	Normal Expanded Wide

(1) Install jumper on header pins 2 and 3.

(2) Install jumper on header pins 1 and 2.

4.6 MEMORY

4.6.1 Memory Types and Sockets

The EVB has footprints for two SRAM sockets (U4, U6A) and two ROM sockets (U7, U9A). The ROM sockets hold memory for D-Bug12, the EVB operating firmware, or for user programs. The SRAM sockets hold memory for user data or programs. The 8-bit memory arrangement allows MCU operation in both single-byte and double-byte modes. The RAM and ROM footprints support different memory device types (SRAM, EPROM, and EEPROM) and sizes (28- and 32-pin, 8 to 512 Kbytes, 300 or 600-mil spacing). Figure 4-1 shows how the external memory sockets are used.

Table 3-5 depicts the EVB's default memory usage. Note that the map is valid only for the factory-supplied memory configuration.

Note that the user-available area in factory-supplied EPROM requires that the ROM chips be reprogrammed with the custom code. For more information, refer to **Appendix E Customizing the EPROMs**.

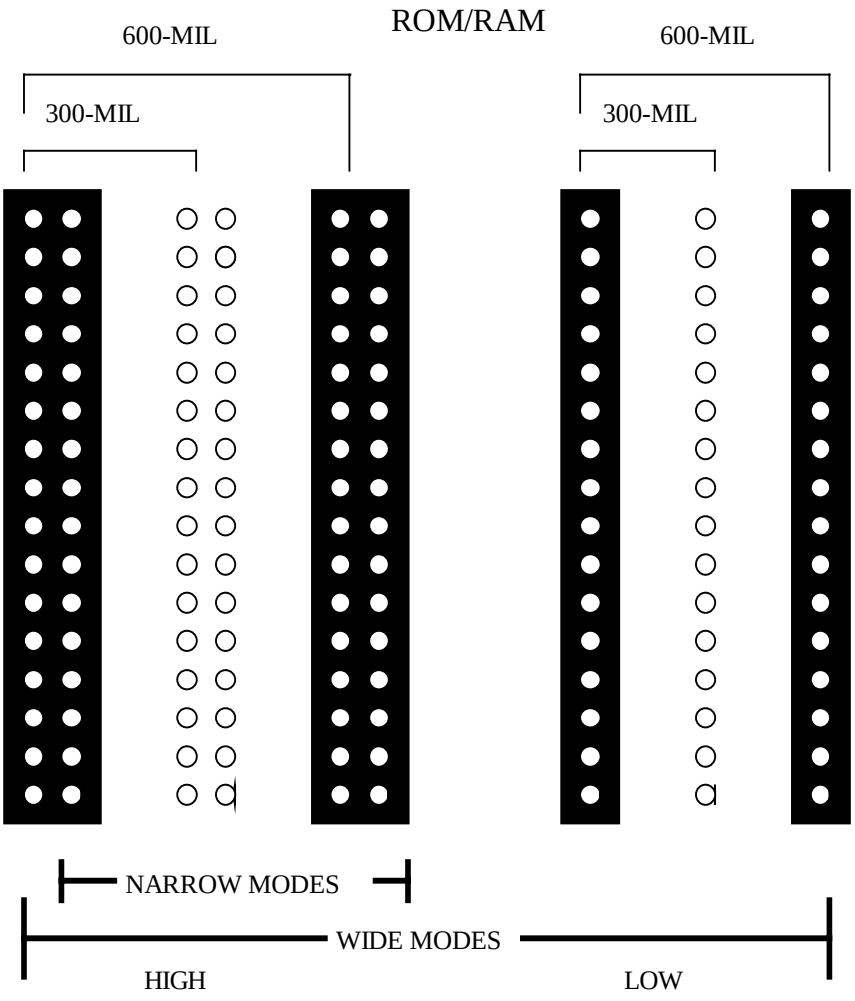


Figure 4-1. Memory Sockets Configuration

Because the EVB is factory-configured for the MCU's Normal Expanded Wide mode, the two RAM and the two ROM sockets are populated with 8-bit memory devices. Only the 600-mil footprints are populated with sockets. There are two RAM and six ROM jumper headers that allow configuration of the memory sockets for use with various types and sizes of memory. These headers are preset for the factory-supplied memories. The default and optional settings are described in Table 4-1. Table 4-3 provides information about the supplied memories.

Table 4-3. EVB Memories Supplied

Type	EPROM	SRAM
Manufacturer	Atmel	Dallas
Part Number	AT27LV256R-20PC	DS2064
Size	256K bits (32K x 8)	64K bits (8K x 8)
Package Width	600 MIL	600 mil
Pin Count	28 pin	28 pin
Power Supply	+3.0 to +5.5 Vdc	+2.7 to +5.5 Vdc
Access Times	200 ns	150 ns @ 5V, 300 ns @ 3V
Wait States Required (E-clock stretches)	1	1

4.6.2 Chip Selects

Header W11 connects an MCU chip select signal to memory devices in the ROM (U7, U9A, U9B) and RAM (U4, U6A, U6B) sockets. Pins in columns 1 and 2 determine the chip select used for memory devices in ROM sockets. Pins in columns 2 and 3 determine the chip select used for memory devices in RAM sockets.

Figure 4-2 shows the W11 jumper settings for the factory-default memory configuration. The illustration demonstrates the correct settings for CSP0* to serve as the ROM chip select and CSD* to serve as the RAM chip select.

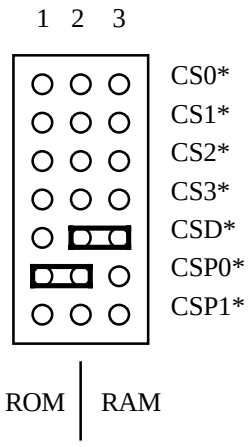


Figure 4-2. Chip Select Header

4.6.3 Glue Logic

Glue logic is required for the MCU to operate with 8-bit memory devices in Wide Expanded modes. It is not needed in Narrow Expanded modes. The EVB allows either an OR gate (U3 — factory-supplied) or a PAL array (U2 — optional, not populated) to serve as the glue logic. Figure 4-3 shows the circuitry for the ROM and RAM logic.

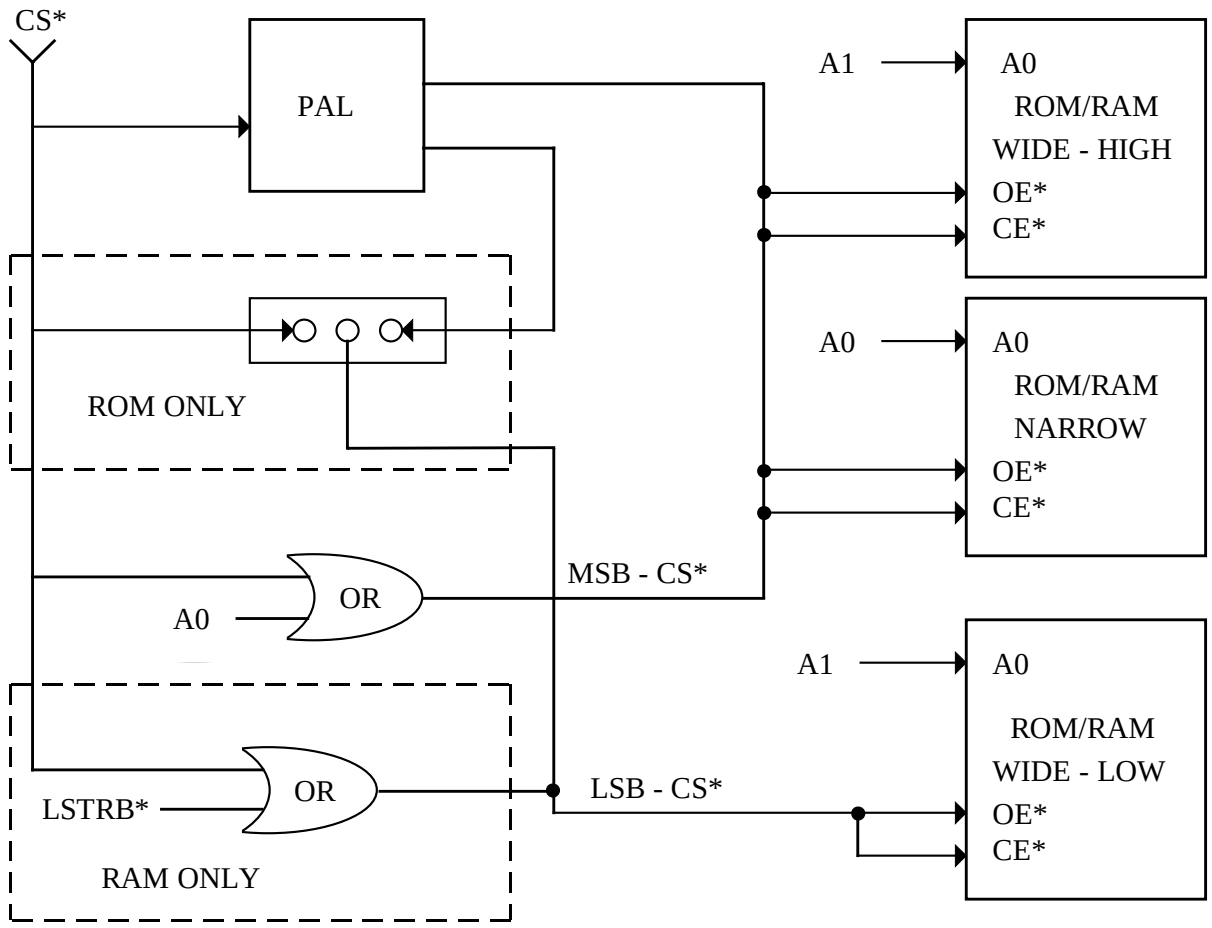


Figure 4-3. RAM/ROM Logic Diagram

4.7 CLOCK CIRCUITRY

The EVB comes with a 16-MHz crystal oscillator installed in a 14-pin DIP socket (XY2). The socket wiring allows the use of various types of oscillator packages. Additionally, there is ancillary circuitry that includes a footprint for a discrete crystal (Y1). This flexible arrangement facilitates the construction of custom oscillators. When designing a custom oscillator, refer to the EVB schematic diagram to locate the applicable components and the headers that must be changed.

An external clock input can be supplied to the MCU's EXTAL by installing a right-angle BNC connector in footprint J7. Refer to the EVB schematic diagram to locate the headers that must be changed.

4.8 PHASE-LOCKED LOOP (PLL)

The PLL can be used to run the MCU on a time base that differs from the clock frequency. To alter the time base, capacitors must be installed between the MCU's XFC pin and the PLL's ground reference, VSS_{PLL}. Connection points E4, E5, E6, E7, E8, and E9 provide space for these capacitors. Header footprint W37 connects the XFC pin to the capacitors.

For more information, refer to the EVB schematic diagram. More detailed information on the operation of the PLL is found in the *MC68HC812A4 Technical Summary.*

4.9 RESET

The reset circuit includes a pull-up resistor, debounce capacitor, and optional connection to an installed undervoltage sensing device (U1, as described in section 4.10). The reset circuit drives the MCU's RESET* pin directly.

4.10 LOW-VOLTAGE INHIBIT

Low voltage inhibition (LVI) uses a Motorola undervoltage sensing device (U1) to automatically drive the MCU's RESET* pin low whenever V_{DD} is below legal limits (2.8 Vdc typical). This prevents the accidental corruption of EEPROM data if the power-supply voltage should drop below the allowable level. Header W1 allows for the disconnection of the LVI circuit.

4.11 ANALOG-TO-DIGITAL (A/D) CONVERTER

The MCU's A/D converter is fully documented in the *MC68HC812A4 Technical Summary.*

Note that two of the A/D bus lines, PAD0 and PAD1, are used by the EVB and D-Bug12 for configuration purposes. These lines are not available for A/D usage in the factory-default configuration.

The accuracy of the A/D converter can be increased by supplying the MCU's A/D circuitry with the same supply voltages used by the target hardware. These supply lines (V_{DDA} and V_{SSA}) and the associated A/D reference voltages (V_{RH} and V_{RL}) can be isolated from the EVB's power bus with cut-trace footprints W15, W16, W17, and W18. Refer to the EVB schematic diagram for details.

4.12 BACKGROUND DEBUG MODE (BDM) INTERFACE

The MCU's serial BDM interface can be accessed through J5, a 2x3 header. The pin assignments are shown in Table 4-4.

Note that the BDM interface requires a development tool such as Motorola's Serial Debug Interface. For more information, refer to Appendix F and to the *Motorola Serial Debug Interface User's Manual*.

Table 4-4. BDM Connector J5 Pin Assignments

Pin Number	Description
1	BKGD
2	V_{SS}
3	no connection
4	RESET*
5	no connection
6	V_{DD}

4.13 PROTOTYPE AREA

The EVB's prototype area allows construction of custom I/O circuitry that can be connected to the MCU's I/O lines through connectors J8 and J9. This 2-inch by 8-inch area is a grid of holes (79 by 20) on 1/10-inch centers. This spacing accommodates most sockets, headers, and device packages..

Figure 4-4 shows the component-side view of the prototype area. Ground (V_{SS}) connections are provided along the three outboard peripheries, with three loop-style test points for connecting clips or probes. Vdc (V_{DD}) connections are provided along the inboard periphery.

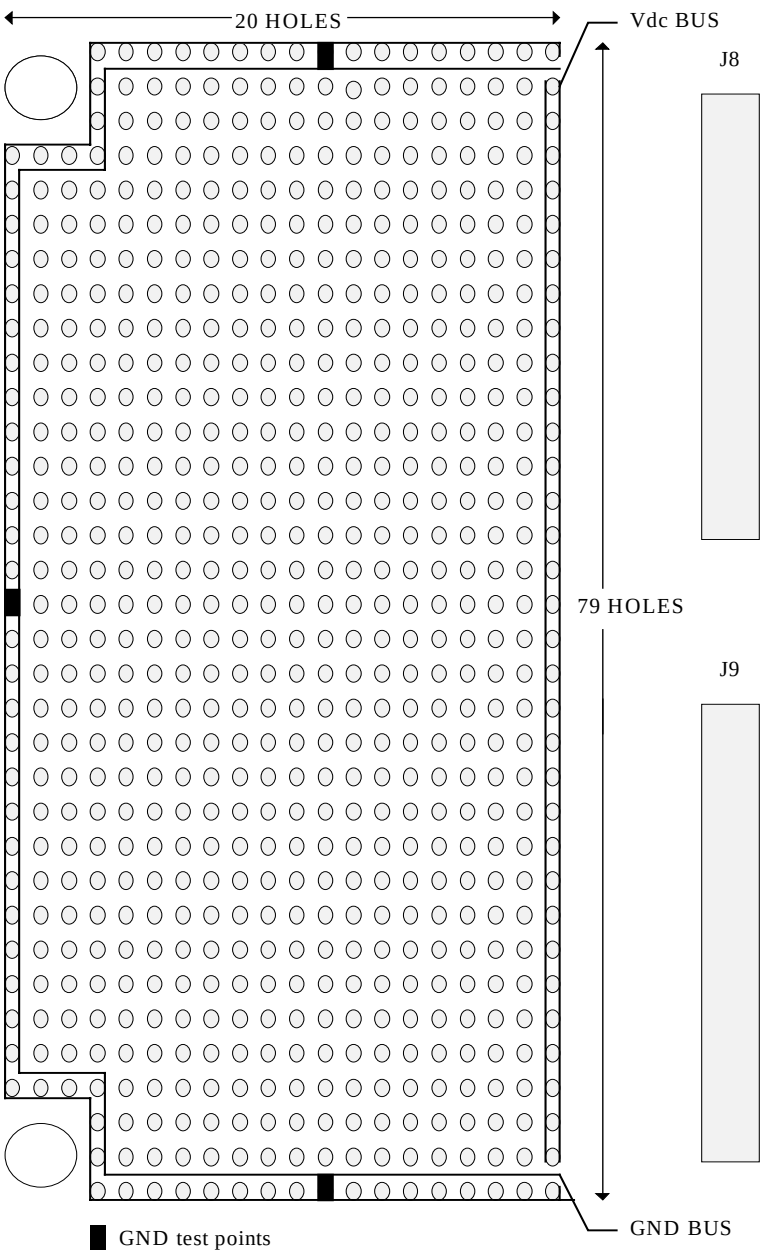


Figure 4-4. Prototype Area (Component-Side View)

4.14 MCU CONNECTORS

Two 2x30 pin header connectors, J8 and J9, provide access to the MCU's I/0 and bus lines. These connectors are located adjacent to the prototype area for use as described in section 4.13. They also provide connection points for instrumentation probes and interfaces to target hardware. Figure 4-5 and Figure 4-6 depict the pin assignments for J8 and J9. Table 4-5 and Table 4-6 provide descriptions of the signals.

Note that the EXTAL, XFC, and XTAL signals are not directly connected to these headers due to impedance considerations. Header footprints W37, W38, and W39 can be used to make these connections.

PJ6	1		2	PJ7
PJ4	3		4	PJ5
PJ2	5		6	PJ3
PJ0	7		8	PJ1
VSSEX0	9		10	VDDEX0
PG4	11		12	PG5
PG2	13		14	PG3
PG0	15		16	PG1
VSSI	17		18	VDDI
BKGD	19		20	NC
PC6	21		22	PC7
PC4	23		24	PC5
PC2	25		26	PC3
PC0	27		28	PC1
PD6	29		30	PD7
PD4	31		32	PD5
PD2	33		34	PD3
PD0	35		36	PD1
PE6	37		38	PE7
PE4	39		40	PE5
PE2	41		42	PE3
PE0	43		44	PE1
NC	45		46	NC
RESET*	47		48	XFC
VSSPLL	49		50	VDDPLL
XTAL	51		52	EXTAL
PB6	53		54	PB7
PB4	55		56	PB5
PB2	57		58	PB3
PB0	59		60	PB1

Figure 4-5. MCU Connector J8 (Component-Side View)

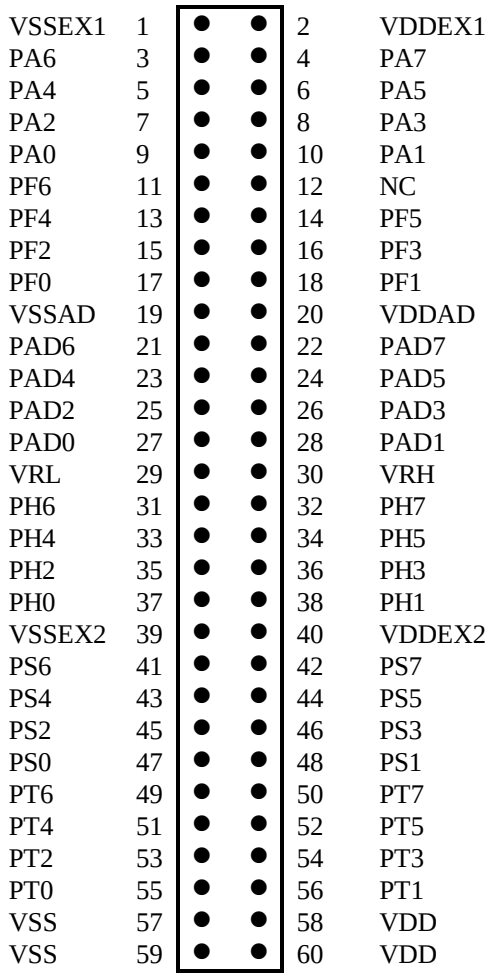


Figure 4-6. MCU Connector J9 (Component-Side View)

Table 4-5. MCU Connector J8 Pin Assignments

Pin Number	Signal Mnemonic	Signal Name And Description
1 2 3 4 5 6 7 8	PJ6/KWUJ6 PJ7/KWUJ7 PJ4/KWUJ4 PJ5/KWUJ5 PJ2/KWUJ2 PJ3/KWUJ3 PJ0/KWUJ0 PJ1/KWUJ1	PORT J (bits 0-7) — general purpose I/O or key wake-up
9 10	VSSEX0 VDDEX0	VSSX/VDDX — external V_{SS} and V_{DD} connections
11 12 13 14 15 16	PG4/A20 PG5/A21 PG2/A18 PG3/A19 PG0/A16 PG1/A17	PORT G (bits 0-5) — general purpose I/O or memory expansion lines
17 18	VSSI VDDI	VSSI/VDDI — internal V_{SS} and V_{DD} connections for the MCU
19	BKGD	BACKGROUND — an I/O line dedicated to the background debug function. If it is a zero out of reset then the MCU is in special mode. This pin can be used for bi-directional communications with the MCU.
20	NC	not connected
21 22 23 24 25 26 27 28	PC6/D14/D6 PC7/D15/D7 PC4D12/D4 PC5/D13/D5 PC2/D10/D2 PC3/D11/D3 PC0/D8/D0 PC1/D9/D1	PORT C (bits 0-7) — general purpose I/O or data bus
29 30 31 32 33 34 35 36	PD6/D6/KWUD6 PD7/D7/KWUD7 PD4/D4/KWUD4 PD5/D5/KWUD5 PD2/D2/KWUD2 PD3/D3/KWUD3 PD0/D0/KWUD0 PD1/D1/KWUD1	PORT D (bits 0-7) — general purpose I/O, data bus, or key wake-up

Table 4-5. MCU Connector J8 Pin Assignments (continued)

Pin Number	Signal Mnemonic	Signal Name And Description
37 38 39 40 41 42 43 44	PE6/MODB/IPIPE1 PE7/ARSIE PE4/E PE5/MODA/IPIPE0 PE2/RW* PE3/LSTRB* PE0/XIRQ* PE1/IRQ*	PORT E (bits 0-7) — general purpose I/O or external signals such as mode select, auxiliary reset, E clock, read/write, strobe low, XIRQ, and IRQ
45 46	NC NC	not connected
47	RESET*	Reset — active-low bi-directional control line used to initialize the MCU
48	XFC	XFC — optional filter-capacitor connection for PLL circuit
49 50	VSSPLL VDDPLL	VSSPLL/VDDPLL — V_{SS} and V_{DD} connections for the PLL circuit.
51	XTAL	CRYSTAL OUTPUT — crystal oscillator output
52	EXTAL	EXTERNAL CLOCK INPUT — crystal oscillator input. The frequency applied to this pin must be twice the desired bus speed.
53 54 55 56 57 58 59 60	PB6/A6 PB7/A7 PB4/A4 PB5/A5 PB2/A2 PB3/A3 PB0/A0 PB1/A1	PORT B (bits 0-7) — general purpose I/O or low byte address bus

Table 4-6. MCU Connector J9 Pin Assignments

Pin Number	Signal Mnemonic	Signal Name And Description
1 2	VSSEX1 VDDEX1	VSSX/VDDX — external V_{SS} and V_{DD} connections
3 4 5 6 7 8 9 10	PA6/A14 PA7/A15 PA4/A12 PA5/A13 PA2/A10 PA3/A11 PA0/A8 PA1/A9	PORT A (bits 0-7) — general purpose I/O or high byte address bus
11	PF6/CSP1*	PORT F (bit 6) — general purpose I/O or chip select
12	NC	not connected
13 14 15 16 17 18	PF4/CSD* PF5/CSP0* PF2/CS2* PF3/CS3* PF0/CS0* PF1/CS1*	PORT F (bits 0-5) — general purpose I/O port or chip selects
19 20	VSSAD VDDAD	VSSAD/VDDAD — V_{SS} and V_{DD} connections for the MCU's A/D converter
21 22 23 24 25 26 27 28	PAD6 PAD7 PAD4 PAD5 PAD2 PAD3 PAD0 PAD1	PORT AD — A/D converter channel or general purpose I/O
29 30	VRL VRH	VOLTAGE REFERENCE, LOW and HIGH — reference voltages for the MCU's A/D converter. These can improve the accuracy of A/D conversions.
31 32 33 34 35 36 37 38	PH6/KWUH6 PH7/KWUH7 PH4/KWUH4 PH5/KWUH5 PH2/KWUH2 PH3/KWUH3 PH0/KWUH0 PH1/KWUH1	PORT H (bits 0-7) — general purpose I/O or key wake-up
39 340	VSSEX2 VDDEX2	VSSX/VDDX — external V_{SS} and V_{DD} connections

Table 4-6. MCU Connector J9 Pin Assignments (continued)

Pin Number	Signal Mnemonic	Signal Name And Description
41 42 43 44 45 46 47 48	PS6/SCK PS7/SS* PS4/MISO PS5/MOSI PS2/RXD1 PS3/TXD1 PS0/RXD0 PS1/TXD0	PORT S (bits 0-7) — general purpose I/O or Multiple Serial Interface (MSI) lines. The MSI lines consist of serial peripheral and serial communication interfaces. The signal functions are serial clock, slave select, master in/slave out, master out/slave in, receiver data input, and transmitter data out.
49 50 51 52 53 54 55 56	PT6/IOC6 PT7/IOC7/PAIN PT4/IOC4 PT5/IOC5 PT2/IOC2 PT3/IOC3 PT0/IOC0 PT1/IOC1	PORT T (bits 0-7) — general purpose I/O or timer lines
57 58 59 60	VSS VDD VSS VDD	VSS/VDD — EVB system return (V_{SS}) and power (V_{DD})

APPENDIX A
S-RECORD FORMAT

DESCRIPTION

The S-record format for output modules was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. The transportation process can thus be visually monitored and the S-records can be more easily edited.

S-RECORD CONTENT

When viewed by the user, S-records are essentially character strings made of several fields that identify the record type, record length, memory address, code/data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first character represents the high-order 4 bits, and the second represents the low-order 4 bits of the byte.

The 5 fields that comprise an S-record are shown below:

TYPE	RECORD LENGTH	ADDRESS	CODE/DATA	CHECKSUM

The S-Record fields are composed as follows:

Field	Printable Characters	Contents
Type	2	S-record type - S0, S1, etc.
Record length	2	The count of the character pairs in the record, excluding the type and record length.
Address	4, 6, or 8	The 2-, 3-, or 4-byte address at which the data field is to be loaded into memory.
Code/data	0-2n	From 0 to n bytes of executable code, memory-loadable data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S-record).
Checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by some time-sharing systems.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

S-RECORD TYPES

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. The various Motorola upload, download, and other record transportation control programs, as well as cross assemblers, linkers, and other file-creating or debugging programs, utilize only those S-records that serve the purpose of the program. For specific information on which S-records are supported by a particular program, the user manual for that program must be consulted.

NOTE

D-Bug12 supports only the S1 and S9 records. All data before the first S1 record is ignored. Thereafter, all records must be S1 type until the S9 record terminates data transfer.

An S-record format module may contain S-records of the following types:

S0	The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. The address field is normally zeroes.
S1	A record containing code/data and the 2-byte address at which the code/data is to reside.

S2-S8	Not applicable to EVB.
S9	A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. If not specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. Normally, only one header record is used, although it is possible for multiple header records to occur.

S-RECORD EXAMPLE

Shown below is a typical S-record format module, as printed or displayed:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The above module consists of an S0 header record, four S1 code/data records, and an S9 termination record.

The S0 header record is comprised of the following character pairs:

S0	S-record type S0, indicating a header record.
06	Hexadecimal 06 (decimal 6), indicating six character pairs (or ASCII bytes) follow.
00 00	Four-character 2-byte address field, zeroes.
48 44 52	ASCII H, D, and R - "HDR".
1B	Checksum of S0 record.

The first S1 code/data record is explained as follows:

S1	S-record type S1, indicating a code/data record to be loaded/verified at a 2-byte address.
13	Hexadecimal 13 (decimal 19), indicating 19 character pairs, representing 19 bytes of binary data, follow.
00 00	Four-character 2-byte address field; hexadecimal address 0000, indicates location where the following data is to be loaded.

The next 16 character pairs are the ASCII bytes of the actual program code/data. In this assembly language example, the hexadecimal opcodes of the program are written in sequence in the

code/data fields of the S1 records:

```
Opcode          Instruction
28 5F           BHCC    $0161
24 5F           BCC     $0163
22 12           BHI     $0118
22 6A           BHI     $0172
00 04 24        BRSET   0,$04,$012F
29 00           BHCS    $010D
08 23 7C        BRSET   4,$23,$018C
                .       (Balance of this code is continued in the
                .       code/data fields of the remaining  S1
                .       records, and stored in memory location
                        0010, etc.)

2A                      Checksum of the first S1 record.
```

The second and third S1 code/data records each also contain $13 (19) character pairs and are ended with checksums 13 and 52, respectively. The fourth S1 code/data record contains 07 character pairs and has a checksum of 92.

The S9 termination record is explained as follows:

S9	S-record type S9, indicating a termination record.
03	Hexadecimal 03, indicating three character pairs (3 bytes) follow.
00 00	Four-character 2-byte address field, zeroes.
FC	Checksum of S9 record.

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted. For example, the first S1 record above is sent as shown below.

TYPE				LENGTH				ADDRESS								CODE/DATA										CHECKSUM			
S		1		1		3		0		0		0		0		2		8		5		F		•••		2		A	
5	3	3	1	3	1	3	3	3	0	3	0	3	0	3	0	3	2	3	8	3	5	4	6	•••		3	2	4	1
0101	0011	0011	0001	0011	0001	0011	0011	0011	0000	0011	0000	0011	0000	0011	0000	0011	0010	0011	1000	0011	0101	0100	0110	•••		0011	0010	0100	0001

APPENDIX B
COMMUNICATIONS PROGRAM EXAMPLES

INTRODUCTION

In all of these examples, first follow the EVB startup procedure in section 3.1. When the startup procedure calls for setting up the host computer's communications program for terminal emulation, follow the steps in the examples.

Keyboard entries are illustrated in this appendix using the following conventions:

<ENTER>	Press the keyboard's Enter, Carriage Return, or Return key.
<ALT-P>	While holding down the ALTERNATE key, press the P key.
<CTL-\>	While holding down the CONTROL key, press the backslash key.
<filename>	Supply the appropriate file name when required.

The stepwise procedures in this appendix are as accurate as possible. However, it is not feasible to document all of the communications programs that are available or to guarantee that a newer revision of a program behaves in exactly the same way as the version used to develop the procedure. For this reason, the steps are as generic as possible in their descriptions. They can thus serve as guidelines for programs not exemplified in this manual. *Always consult the documentation for the program being used.*

PROCOMM FOR DOS — IBM PC

Setup

To set up Procomm using DOS on an IBM-compatible PC for use as the EVB terminal, first refer to section 3.1 for the EVB startup procedure, which is inter-related with this example. Then follow these steps:

1. At the DOS prompt, Invoke the Procomm program by typing:

   ```
   PROCOMM<RETURN>
   ```

2. Enter the Setup menu by pressing <ALT-S>.
3. From the TERMINAL SETUP submenu, select the following:

Terminal emulation	WYSE 100
Duplex	FULL

Flow control	NONE
CR translation (in)	CR
CR translation (out)	CR
BS translation	DEST
BS key definition	BS
Line wrap	OFF
Scroll	ON
Break Length (ms)	350
Enquiry (CTRL-E)	OFF

4. From the ASCII TRANSFER SETUP submenu, select the following:

Echo locally	YES
Expand blank lines	YES
Pace character	0 (ASCII)
Character pacing	25 (1/1000th sec)
Line pacing	10 (1/10th sec)
CR translation	NONE
LF translation	NONE

5. Enter the Line Settings menu by pressing <ALT-P>. Select the following:

baud rate	9600 (or the customized EVB setting)
data bits	8
stop bits	1
parity	none
COM port	the host port used as the EVB terminal interface

6. Reset the EVB by pressing S1 or by activating the appropriate custom reset circuitry.
7. Press <ENTER>. The D-Bug12 prompt should appear on the display. Continue with the startup procedure in section 3.1.

S-Record Transfers to EVB Memory

To load an S-Record file from the host computer into EVB memory using Procomm on an IBM-compatible host computer, first verify that the host is correctly configured and operating as the EVB terminal. Then follow these steps:

1. At the D-Bug12 prompt, enter the LOAD or VERF command with any parameters.

2. Instruct Procomm to send the S-Record file by pressing the <Page Up> key. Follow the onscreen instructions to select the S-Record file for transfer, using ASCII transfer protocol.

Upon completion of the S-Record file transfer, the D-Bug12prompt is displayed.

KERMIT FOR DOS — IBM PC

Setup

To set up Kermit using DOS on an IBM-compatible PC for use as the EVB terminal, first refer to section 3.1 for the EVB startup procedure, which is inter-related with this example. Then follow these steps:

1. At the DOS prompt, invoke Kermit by typing:

   ```
   kermit<ENTER>
   ```

2. Set the baud rate to 9600 (or the customized EVB setting) by typing:

   ```
   set baud 9600<ENTER>
   ```

3. Connect to the EVB by typing:

   ```
   connect<ENTER>
   ```

4. Reset the EVB by pressing S1 or by activating the appropriate custom reset circuitry. The D-Bug12 prompt should appear on the display. Continue with the startup procedure in section 3.1.

S-Record Transfers to EVB Memory

To load an S-Record file from the host computer into EVB memory using Kermit on an IBM-compatible host computer, first verify that the host is correctly configured and operating as the EVB terminal. Then follow these steps:

1. At the D-Bug12 prompt, enter the LOAD or VERF command with any parameters.
2. “Escape” from the D-Bug12 prompt and start the Kermit file transfer by typing:

   ```
   <CTL-]>c
   push<ENTER>
   type <filename> > com1<ENTER>
   ```

Upon completion of the S-Record file transfer, the D-Bug12 prompt is displayed.

KERMIT — SUN WORKSTATION

Setup

To set up Kermit on the Sun Workstation for use as the EVB terminal, first refer to section **Error! Reference source not found.** for the EVB startup procedure, which is inter-related with this example. Then follow these steps:

1. In a shell window, invoke Kermit by typing:

   ```
   kermit<ENTER>
   ```

2. Set the serial port to the one in use for the EVB (ttya, ttyb, etc.) by typing:

   ```
   set line /dev/ttya<ENTER>
   ```

3. Set the baud rate to 9600 (or the customized EVB setting) by typing:

   ```
   set speed 9600<ENTER>
   ```

4. Connect to the EVB by typing:

   ```
   connect<ENTER>
   ```

5. Reset the EVB by pressing S1 or by activating the appropriate custom reset circuitry. The D-Bug12 prompt should appear on the display. Continue with the startup procedure in section **Error! Reference source not found.**.

S-Record Transfers to EVB Memory

To load an S-Record file from the host computer into EVB memory using Kermit on a Sun Workstation, first verify that the host is correctly configured and operating as the EVB terminal. Then follow these steps:

1. In the shell window being used for the EVB terminal interface, at the D-Bug12 prompt, enter the LOAD or VERF command with any parameters.
2. Open a shell window separate from the one being used for the EVB terminal interface. In this window, type:

   ```
   cat <filename> > /dev/ttya<ENTER>
   ```

Upon completion of the S-Record file transfer, the D-Bug12 prompt is displayed in the shell window being used for the EVB terminal interface.

MACTERMINAL — APPLE MACINTOSH

Setup

To set up MacTerminal on an Apple MacIntosh computer for use as the EVB terminal, first refer to section 3.1 for the EVB startup procedure, which is inter-related with this example. Then follow these steps:

1. Select the following from the Terminal Settings menu:

Terminal:	TTY
Cursor Shape:	Underline
Line Width:	80 Columns
Select:	On Line
	Auto Repeat
Click on:	OK

2. Select the following from the Compatibility Settings menu:

Baud Rate:	9600 (or the customized EVB setting)
Bits per Character:	8 Bits
Parity:	None
Handshake:	None
Connection:	Modem or Another Computer
Connection Port:	Modem or Printer
Click on:	OK

3. Reset the EVB by pressing S1 or by activating the appropriate custom reset circuitry.
4. Press <ENTER>. The D-Bug12 prompt should appear on the display. Continue with the startup procedure in section 3.1.

S-Record Transfers to EVB Memory

To load an S-Record file from the host computer into EVB memory using MacTerminal, first verify that the host is correctly configured and operating as the EVB terminal. Then follow these steps:

1. At the D-Bug12 prompt, enter the LOAD or VERF command with any parameters.
2. From the MacIntosh File menu, select Send File - ASCII.
3. From the dialog box, select the S-Record file to be transferred.

4. Click on Send.

NOTES

1. S-Records are not displayed during the file transfer.
2. Following the file transfer, MacTerminal sends a carriage return-line feed pair, which D-Bug12 interprets as an erroneous command. To return to the D-Bug12 prompt, reset the EVB.

RED RYDER — APPLE MACINTOSH

Setup

To set up Red Ryder on an Apple MacIntosh computer for use as the EVB terminal, first refer to section 3.1 for the EVB startup procedure, which is inter-related with this example. Then follow these steps:

1. Launch the Red Ryder program.
2. Set up the Red Ryder parameters as follows:

   9600 baud (or the customized EVB setting)

   8 data bits

   1 stop bit

   no parity

   full duplex
3. Reset the EVB by pressing S1 or by activating the appropriate custom reset circuitry.
4. Press <ENTER>. The D-Bug12 prompt should appear on the display. Continue with the startup procedure in section 3.1.

S-Record Transfers to EVB Memory

To load an S-Record file from the host computer into EVB memory using Red Ryder, first verify that the host is correctly configured and operating as the EVB terminal. Then follow these steps:

1. At the D-Bug12 prompt, enter the LOAD or VERF command with any parameters.
2. From the MacIntosh File menu, select Send File - ASCII.
3. From the dialog box, select the S-Record file to be transferred.
4. Click on Send.

NOTE

S-Records are not displayed during the file transfer.

Upon completion of the S-Record file transfer, the D-Bug12 prompt is displayed.

APPENDIX C

D-BUG12 STARTUP CODE

The D-Bug12 startup code is located in the EPROMs, U7 and U9A, in the address range $FD80 to $FDFF, as shown in Table 3-5.

To customize this startup code, it is necessary to reprogram the EPROMs. For more information, refer to Appendix E, Customizing the EPROMs.

The following D-Bug12 startup code is distilled from the source listing for clarity. To assemble the startup code for programming into the EPROMs, the .DEFINEs must be included ahead of the code listed below. These are available on the Internet at http://www.mot.com/m68hc12.

```
                        opt   lis               ; assembler directive to turn
                                                ; listing on

0A00        MonRAMStart equ   $0A00
0200        MonRAMSize  equ   $0200

0800        RAM_START   equ   $0800

0400        RAMSize     equ   $0400

0C00        STACKTOP    equ   RAM_START+RAMSize ; stack at top of int RAM

1000        EE_START    equ   $1000             ; 4K EEPROM located here out
                                                ; of reset(in expanded modes)
FD80                    org   $fd80

            ;*************************************************************
            ;
            ; INITIALIZATION
            ;
            ; Initialization code for the M68HC12A4EVB D-Bug12 monitor program
            ;*************************************************************

FD80        CODE_START:

            ;     set PortE bit 7 to an output to eliminate possible noise
            ;     problems associated with unterminated input pins.
```

```
FD80 4C0980             bset  DDRE,80h          ; set the data direction to
                                                ; configure PortE, bit 7 as an
                                                ; output.
FD83 4C0880             bset  PORTE,80h         ; set PortE, bit 7 to logic 1.

FD86 CF0C00             lds   #STACKTOP         ; initialize D-Bug12 stack
                                                ; pointer
FD89 4F6F0103           brclr PORTAD,01h,DEBUG12; if bit 0 of A/D port is 1,
FD8D 061000             jmp   EE_START          ; then jump to the start of
                                                ; internal EEPROM
                                                ; otherwise, remain in D-Bug12
FD90        DEBUG12:

            ;     Clear all monitor RAM to start from a known state

FD90 CE0A00             ldx   #MonRAMStart
FD93 6930   ClrRAM:     clr   1,x+              ; clear one and inc pointer
FD95 8E0C00             cpx   #MonRAMStart+MonRAMSize
FD98 26F9               bne   ClrRAM            ; loop till RAM clear

            ;     Enable pipe signals, E, low strobe and read/write in port E
            ;     PIPOE, NECLK, LSTRE and RDWE are write once in normal modes
            ;     PEAR  [ARSIE:CDLTE :PIPOE :NECLK !LSTRE : RDWE :  0   :  0   ]$0A

FD9A 862C               ldaa  #$2c              ; prevent later protection
                                                ; lock
FD9C 5A0A               staa  PEAR              ; PROTLK is write-once

            ;     Without changing modes, enable internal visibility
            ;     MODE  [SMODN: MODB : MODA : ESTR ! IVIS :  0   : EMD  : EME  ]$0B

FD9E 4C0B08             bset  MODE,$08          ; set IVIS

            ;     Disable the COP watchdog by CR2:CR1:CR0 = 0:0:0
            ;     COPCTL = $07 when reset in normal modes
            ;     FCME and CRx bits are write once in normal modes
            ;     COPCTL[ CME : FCME : FCM  : FCOP ! DISR : CR2  : CR1  : CR0  ]$16

FDA1 790016             clr   COPCTL            ; disable watchdog

            ;     Enable Program chip select 0 and Data chip select
            ;     CSCTL0 = $20 after reset (CSP0 on others off)
            ;     also set data chip select to cover $0000-7FFF (will mirror
            ;     to fill space)
            ;     internal resources have higher priority in case of overlaps
            ;
            ;     CSCTL0[  0  :CSP1E :CSP0E : CSDE ! CS3E : CS2E : CS1E : CS0E ]$3C
            ;     CSCTL1[  0  :CSP1FL:CSPA21:CSDHF !CS3EP :  0   :  0   :  0   ]$3D

FDA4 8630               ldaa  #$30
FDA6 5A3C               staa  CSCTL0            ; CSP0E and CSDE on
FDA8 8610               ldaa  #$10
FDAA 5A3D               staa  CSCTL1            ; CSD to cover $0000-7FFF
```

```
            ;     Set stretch for CSP0 and CSD to 1 extra E-speed cycle per
            ;     access (to accomodate slower external RAM and EPROM)
            ;
            ;     CSSTR0[  0  :  0   :SRP1A :SRP1B !SRP0A :SRP0B :STRDA :STRDB ]$3E

FDAC 8605             ldaa  #$05
FDAE 5A3E             staa  CSSTR0            ; CSP0E and CSDE on

            ;     Enable EEPROM so D-Bug12 can program/erase bytes
            ;     EEMCR [  1  :  1   :  1   :  1   !  1   :  1   :PROTLK: EERC ]$F0
            ;     BPROT [  1  :BPROT6:BPROT5:BPROT4!BPROT3:BPROT2:BPROT1:BPROT0]$F1

FDB0 86FC             ldaa  #$fc              ; prevent later protection
                                              ; lock
FDB2 5AF0             staa  EEMCR             ; PROTLK is write-once
FDB4 7900F1           clr   BPROT             ; allow EE program and erase

FDB7 CEFE00           ldx   #$fe00            ; point to the table of user
                                              ; accessible routines.
FDBA 05E30000         jmp   [0,x]             ; the first entry is a pointer
                                              ; to main. GO.........


            ;     The following subroutine produces a delay of approximately
            ;     20 mS, based on the following conditions:

            ;     1.) An 8.00 MHz E-clock
            ;     2.) Subroutine located in external EPROM - selected by CSP0
            ;     3.) CSP0 programmed for 1 E-clock stretch
            ;
            ;     This routine is called by D-Bug12's WriteEEByte() function
            ;     through a pointer stored in the Customization Data Table.

FDBE        _EEDelay:
FDBE CE2710           ldx   #10000            ; load delay count into x
FDC1 09     DlyLoop:  dex                     ; decrement count
FDC2 26FD             bne   DlyLoop           ; loop till done.
FDC4 3D               rts                     ; return.
```

APPENDIX D

D-BUG12 CUSTOMIZATION DATA

The Customization Data area, located in EPROM from `$FE80 to $FEFF`, allows users to change default data parameters used by D-Bug12. The data contained in this area is described by C data structure. The `CustomData` typedef is shown below. For those unfamiliar with C an assembly language equivalent is also shown. The purpose of each field is explained in the following paragraphs.

```
typedef struct {
               Byte UserCCR;                 /* User CPU Condition Code Register */
               Byte UserB;                   /* User CPU B-accumulator */
               Byte UserA;                   /* User CPU A-accumulator */
               Address UserX;                /* User CPU X-index register */
               Address UserY;                /* User CPU Y-index register */
               Address UserPC;               /* User CPU Program Counter */
               Address UserSP;               /* User CPU Stack Pointer */
               unsigned long SysClk;         /* System Clock frequency (in Hz) */
               Address IOBase;               /* Base address of the I/O registers */
               unsigned int SCIBaudRegVal;   /* Initial SCI BAUD register value */
               Address EEBase;               /* Base address of on-chip EEPROM */
               unsigned int EESize;          /* size of the on-chip EEPROM */
               void (*Delay)(void);          /* pointer to EEPROM program/erase */
                                             /* delay routine */
               int AuxCmdCount;              /* number of commands in the */
                                             /* auxiliary command table */
               CmdTblEntryP AuxCmdTableP;    /* pointer to the auxiliary command */
                                             /* table */
               } CustomData;
```

```
                   org    $FE80
;
CustData           equ    *
UserCCR            dc.b   $90          ; User CPU Condition Code Register
UserB              dc.b   $00          ; User CPU B-accumulator
UserA              dc.b   $00          ; User CPU A-accumulator
UserX              dc.w   $0000        ; User CPU X-index register
UserY              dc.w   $0000        ; User CPU Y-index register
UserPC             dc.w   $0000        ; User CPU Program Counter
UserSP             dc.w   $0A00        ; User CPU Stack Pointer
SysClk             dc.l   8000000      ; System Clock frequency (in Hz)
IOBase             dc.w   $0000        ; Base address of the I/O registers
SCIBaudRegVal dc.w   52           ; Initial SCI BAUD register value
EEBase             dc.w   $1000        ; Base address of the on-chip EEPROM
EESize             dc.w   4096         ; Size of the on-chip EEPROM
EEDelay            dc.w   _EEDELAY     ; Address of EEPROM program/erase delay
                                       ; routine
AuxCmdCount        dc.w   0            ; Number of commands in the auxiliary
                                       ; command table
AuxCmdTableP       dc.w   $0000        ; Pointer to the auxiliary command table
```

Initial User CPU Register Values

The first seven fields in the `CustomData typedef struct` are used to provide default values for the user CPU12 registers. The user CCR value is set to 0x90. This sets the S-bit, disabling the STOP instruction, and the I-bit, inhibiting IRQ interrupts. The X-bit is cleared to allow the use of the XIRQ interrupt as a programmer's abort switch. The user SP value is set to 0x0a00, which is one byte beyond the last on-chip RAM location available to the user. The CPU12 stack pointer points to the last byte pushed onto the stack. All of the other registers contain the value zero.

SysClk Field

The `SysClk` field is used to inform D-Bug12 of the system clock frequency, M. Its value, in Hz, is set to 8,000,000. The E-clock frequency is the same as the system clock frequency, M. `SysClk` is used by the D-Bug12 BAUD command in calculating the new value of the SCI Baud register for the requested baud rate.

NOTE

It is the responsibility of the startup code to perform any actions necessary to set the system clock frequency. D-Bug12 DOES NOT set or change the system clock frequency using the `SysClk` value.

IOBase Field

The `IOBase` field defines the base address of the I/O registers. This address is used by D-Bug12 when accessing the I/O registers associated with the SCI and when programming or erasing the on-chip EEPROM. On the MC68HC812A4 the I/O registers are mappable to any 2k memory space. Therefore, the `IOBase` entry should only be a multiple of 2048. The value of `IOBase` is set to 0x0000 which is the default address of the I/O registers for the MC68HC812A4.

NOTE

It is the responsibility of the startup code to set the base address of the I/O registers. D-Bug12 DOES NOT set or change the I/O register base address.

SCIBaudRegVal Field

The `SCIBaudRegVal` field is used to set the initial baud rate of the SCI used for console I/O by D-Bug12. Note that the value in `SCIBaudRegVal` is written directly to the Baud register of the console SCI. The value is NOT the desired baud rate. The calculation of this value is NOT made by D-Bug12 because of the possibility of an invalid Baud register value. Without a valid Baud register value during SCI initialization, D-Bug12 would have no way to inform the user that a problem existed. Not all combinations of baud rates and system clock frequencies produce a valid Baud register value. The formula used to calculate the Baud register value is:

```
BaudRegVal = MCLK ÷ ( 16 * SCIBaudRate)
```

The initial Baud register value is 52 (0x0034). At a system clock frequency of 8.0 MHz, this sets the communications rate of 9600 baud.

NOTE

Because of the ability to choose either SCI0 or SCI1 for use as the control console, D-Bug12 takes care of initializing the SCI registers. The chosen SCI is set to 8-data bits, 1-start bit, 1-stop bit, and no parity.

EEBase and EESize Fields

The `EEBase` and `EESize` fields are used to describe the base address and range of the M68HC12's on-chip EEPROM. This information is used by D-Bug12's `WriteMem()` function to determine when a byte is being written to the on-chip EEPROM. D-Bug12 then calls its WriteEEByte() function to place the data in the on-chip EEPROM. On the MC68HC812A4 the EEPROM base address is mappable to any 4k memory space. Therefore, the `EEBase` entry should only be a multiple of 0x1000. The value of `EEBase` is set to 0x1000 which is the default base address of the on-chip EEPROM for the MC68HC812A4. The value of `EESize` is also set to 0x1000 (4096) which is the size of the on-chip EEPROM. Setting the value of `EESize` to zero disables the `WriteMem()` function's ability to write to on chip EEPROM.

NOTE

It is the responsibility of the startup code to set the base address of the EEPROM. D-Bug12 DOES NOT set or change the EEPROM base address.

EEPROM Erase/Program Delay Function Pointer Field

The `(void)(* Delay)(void)` field is a function pointer that points to an EEPROM program/erase delay routine. For the MC68HC812A4, the routine should produce a delay of 20 mS before it returns. The delay routine is nothing more than a software delay loop. The subroutine is located in the startup code area of the D-Bug12 EPROM from `$FD80 - $FDFF`. See Appendix C, D-Bug12 Startup Code.

Auxiliary Command Table Entries

The last two entries in this table provide a mechanism to extend the command set of D-Bug12. The `AuxCmdTableP` points to an auxiliary command table, and `AuxCmdCount` contains the number of entries in the auxiliary command table. The table consists of an array of `CmdTblEntry's.` Each `CmdTblEntry` in the auxiliary command table has the following structure:

```
typedef struct {
      const char *CommandStr;                    /* pointer to the  command */
                                                 /*string */
      int (*ExecuteCmd)(int argC, char *argV[]);/* pointer to function that*/
                                                 /* implements the command */
               } CmdTblEntry, * CmdTblEntryP;
```

As the `typedef` shows, the first field is a character pointer pointing to a null terminated character array containing the command name. The command name string *must be in upper case*. The second field, a function pointer, points to a function that implements the new D-Bug12 command. The first parameter to this function is a count of the number of command line arguments that the command line interpreter found on the command line. This count *includes* the command name itself. The command line may contain no more than a total of 10 parameters. The second function parameter is a pointer to an array of `char *`. Each `char *` points to one of the command line parameters parsed by the command line interpreter.

The function implementing the new command can report any error conditions to the user in one of two ways. If the error condition can be described by one of the error messages in the enumerated constant list below, the user defined command should return the appropriate constant. If some other message text needs to be conveyed to the user, the command should communicate the error message directly to the user by using the `printf()` function which is one of the available user callable functions. In this case, the user defined command should return an error code of noErr.

```
enum Error {
            WrongNumArgs = 6,     /* Wrong Number of Arguments */
            BadStartAddress = 7, /* Invalid Starting Address */
            BadEndAddress = 8,   /* Invalid Ending Address */
            StartEndError = 9,   /* Start Address Greater Than End Address */
            BadHexData = 10,     /* Invalid Hex Data */
            DataSizeError = 11,  /* Data Out Of Range */
            NoTargetWrite = 12,  /* Can't Write Target Memory */

           };
```

APPENDIX E
CUSTOMIZING THE EPROMS

The following blocks in the factory-supplied EPROMs can be reprogrammed with user code or D-Bug12 code that has been modified for custom operation:

$8000 - $9FFF — available for user programs

$FD80 - $FDFF — D-Bug12 startup code. See Appendix C.

$FE80 - $FEFF — D-Bug12 customization data. See Appendix D.

$FF00 - $FFBF — available for user programs

Since the EPROMs also contain D-Bug12 and other EVB operating firmware, the factory programming must be retained and burned into the custom chips along with the custom code. The table below maps the EVB's logical addresses (from Table 3-5) to the pin-level physical addresses of U7 and U9A.

Note that the lower half of each EPROM — from $0000 to $3FFF — is unused and is filled with ones. This is necessary because of the chip select, CSP0*, used by the MCU for EPROM access. For more information on this subject, refer to **4.6.2 Chip Selects**.

NOTE

Do not reprogram the factory-supplied EPROMs. Keep them as masters, using expendable chips for new programming.

Physical EPROM Addresses

MCU Logical Address	Data	U9A Physical Address	U7 Physical Address
—	$FF	$0000 - $3FFF	$0000 - $3FFF
$8000 - $9FFE even addresses	custom	$4000 - $4FFF	—
$8001 - $9FFF odd addresses	custom	—	$4000 - $4FFF
$A000 - $FD7E even addresses	factory	$5000 - $7EBF	—
$A001 - $FD7F odd addresses	factory	—	$5000 - $7EBF
$FD80 - $FDFE even addresses	factory or modified	$7EC0 - $7EFF	—
$FD81 - $FDFF odd addresses	factory or modified	—	$7EC0 - $7EFF
$FE00 - $FE7E even addresses	factory	$7F00 - $7F3F	—
$FE01 - $FE7F odd addresses	factory	—	$7F00 - $7F3F
$FE80 - $FEFE even addresses	factory or modified	$7F40 - $7F7F	—
$FE81 - $FEFF odd addresses	factory or modified	—	$7F40 - $7F7F
$FF00 - $FFBE even addresses	custom	$7F80 - $7FBF	—
$FF01 - FFBF odd addresses	custom	—	$7F80 - $7FBF
$FFC0 - $FFFE even addresses	factory	$7FC0 - $7FFF	—
$FFC1 - $FFFF odd addresses	factory	—	$7FC0 - $7FFF

APPENDIX F
SDI CONFIGURATION

To configure the EVB for use with Motorola's Serial Debug Interface (SDI), follow these steps:

1. Remove the jumper on header W11 from CSD*.
2. Move the CSP0* jumper on W11 to pins 2-3.

   Steps 1 and 2 disable the external EPROM and map the CSP0* chip select to external RAM.
3. Remove the jumper from W30.

   Step 3 allows the SDI to drive the MCU's BKGD pin low at reset.
4. Move the jumper on W34 to pins 1-2.
5. Move the jumper on W42 to pins 1-2.

   Steps 4 and 5 place the MCU in Special Single Chip mode.
6. Move the base address of the MCU's on-chip EEPROM from $F000 (the default in Special Single Chip mode) to $1000. To do this, change the data at address $0012 to a value of $11 using the appropriate debugging tool. For MCUdebug, the correct command is:

   ```
   MM 12 11
   ```

   Step 6 must be repeated each time the EVB is reset in this mode, as the EEPROM's base address defaults to $F000 at reset.

Table 4-1 provides full descriptions of these jumper changes. See Figure 4-2 for details of header W11. See Figure 1-1 for header locations on the EVB.

Note that CSP0* covers the address range from $8000 to $FFFF. The 16 Kbytes of RAM appear in the new memory map from $C000 to $FFFF. This SDI memory map is shown in the table below.

This configuration provides the following enhancements when using the SDI:

- The MCU's on-chip RAM, from $0800 to $0BFF, is entirely available for user data.
- Data can be loaded into the vector area, which was reserved under the D-Bug12 operating configuration.

For information on using the SDI, refer to the *Motorola Serial Debug Interface User's Manual*.

SDI Memory Map

Address Range	Description	Location
$0000 - $01FF	CPU registers	on-chip (MCU)
$0800 - $0BFF	user data area	1K on-chip RAM (MCU)
$1000 - $1FFF	user code area	4K on-chip EEPROM (MCU)
$C000 - $FFFF	user code/data area	16K external RAM (U4, U6A)

INDEX

—A—

A/D converter
  description, 4-14
  isolatable power circuits, 4-6, 4-14

—B—

background debug mode (BDM)
  as user interface, 1-6, 1-7, 2-4
  interface connector, J5, 4-15
  MCU mode, 3-26, 4-8
block diagram
  EVB system, 1-4
bulletin boards, 1-9, C-1

—C—

chip select. *See* memory, chip selects
clock
  circuitry, 4-13
  E-clock, 1-5, 2-6, 4-11, C-3
  external input, 4-14
  oscillator chip and socket, 4-13
  speed, 1-7, 4-13
  time base, 4-14
code
  firmware modification, C-1
  generation, 1-6, 3-32
commands, D-Bug12
  <REGISTER NAME> — Modify Register Value, 3-30
  ASM — Assembler/Disassembler, 3-6
  BAUD — Set Baud Rate, 3-9
  BF — Block Fill, 3-10
  BR — Breakpoint Set, 3-11
  BULK — Bulk Erase on-chip EEPROM, 3-12
  CALL — Call Subroutine, 3-13
  GO — Go Execute a User Program, 3-14
  Go Till, 3-15
  HELP — Onscreen Help Summary, 3-16
  LOAD — Load S-Record File, 3-17
  MD — Memory Display, 3-18
  MDW — Memory Display, Word, 3-19
  MM — Memory Modify, 3-20
  MMW — Memory Modify, Word, 3-21
  MOVE — Move Memory Block, 3-22
  NOBR — Remove Breakpoints, 3-23
  RD — Register Display, 3-24
  RM — Register Modify, 3-25
  T — Trace, 3-26
  UPLOAD — Display Memory, S-Record Format, 3-28
  VERF — Verify S-Record File against Memory, 3-29
communications, EVB-host
  baud rate, 2-5, 3-9
  limitations, 3-35
  parameters, 2-4, 2-5
  SCI ports, 2-3, 4-6
  software, 1-7, 2-5, B-1
configuration
  D-Bug12, C-1
  EVB, 2-2
  jumpers, 4-1
  SDI, F-1
connectors
  J1, J2 — SCI1 RS-232C port, 2-3, 4-6
  J3, J4 — SCI0 RS-232C port, 2-3, 4-6
  J5 — BDM interface, 4-15
  J6 — power input, 2-2, 4-6
  J7 — external clock, 4-14
  J8, J9 — MCU access, 1-6, 4-15, 4-17
  locations, 1-3
  types, 4-1
CPU
  instruction translation, 3-6, 3-7
  registers. *See* registers
  type. *See* MCU
crystal. *See* clock
customer support, 1-9

—D—

D-Bug12
  aborting a user program, 3-2
  command set, 3-4, 3-5
  command-line format, 3-3
  commands. *See* commands, D-Bug12
  configuration requirements, 1-5, 1-6, 2-2, 2-4, 4-1
  customization data, D-1
  description, 1-5, 1-6
  generating user code, 1-6, 3-32
  limitations imposed by, 1-7, 3-34
  memory usage, 3-33, 3-34, E-1
  resetting, 3-2
  stack pointer, 3-33
  starting, 3-1
  startup code, C-1
  startup modes, 1-6, 2-2, 3-1, 3-32
  terminal interface, 1-5, 4-6
DS1. *See* power, indicator

—E—

E-clock, 1-5, 2-6, 4-11, C-3
EEPROM. *See also* memory
  starting execution from, 3-32
EPROM. *See* memory
evaluation board. *See* EVB

EVB
block diagram, 1-4
component placement, 1-3
configuring, 2-2, 4-1
description, general, 1-1
description, hardware, 4-1
features, 1-1
firmware. *See* D-Bug12
functional overview, 1-5
operating instructions, 3-1
packing list, 2-1
restrictions on use, 3-34
specifications, 1-8
unpacking, 2-1

—F—

file transfers, 3-17, 3-29, 3-32, B-1
firmware. *See* D-Bug12

—H—

headers
connector, 4-1. *See also* connectors
cut-trace, 4-1
description, 4-1
jumper, 4-1. *See also* jumper settings

—J—

J1, J2 — SCI1 RS-232C port, 2-3, 4-6
J3, J4 — SCI0 RS-232C port, 2-3, 4-6
J5 — BDM interface, 4-15
J6 — power input, 2-2, 4-6
J7 — external clock, 4-14
J8, J9 — MCU access, 1-6, 4-15, 4-17
jumper settings, 1-2, 1-5, 4-1, 4-3

—L—

LED. *See* power, indicator
low voltage inhibit (LVI), 4-14

—M—

M68HC12A4EVB Evaluation Board. *See* EVB
MC68HC812A4 Microcontroller Unit. *See* MCU
MCU
access interface, 1-6, 4-15, 4-17
description, 4-7
isolatable power circuits, 4-6
location, 1-3
modes, 4-7, 4-8, 4-9, 4-10
restrictions on use, 1-6, 3-33, 3-34
socket, 2-1
type, 1-8, 4-7
memory
and MCU modes, 4-7
chip selects, 1-5, 2-6, 4-11, F-1
configurations, 3-33, 4-9, 4-10
customizing the EPROMs, E-1
EEPROM, external, 4-9
EEPROM, on-chip, 1-6, 2-2, 3-12, 3-32, 4-14
EPROM, 1-5, 4-9, E-1
external, 4-9
glue logic, 4-12
limitations, 3-33, 3-34
loading from host computer, 3-32
locations, 1-3, 1-4
map, EPROM, E-2
map, factory default, 3-33, 3-34
map, SDI configuration, F-2
on-chip, 4-7, F-2
programming, 1-6
RAM, 1-4, 2-6, 4-9
ROM, 4-9
sockets, 4-9, 4-10
speed enhancement, 1-4, 2-6
SRAM, 1-4, 2-6, 4-9
usage, 3-33, 4-9
wait states, 1-4, 2-6, 4-11
microcontroller unit. *See* MCU
monitor program. *See* D-Bug12
multiple serial interface (MSI), 4-22

—O—

oscillator. *See* clock

—P—

packing list, 2-1
phase-locked loop (PLL)
description, 4-14
isolatable power circuit, 4-6
power
distribution, 4-6, 4-15, 4-16, 4-17
indicator,description, 4-6
indicator,location, 1-3
input circuit and protection, 4-6
input connector, J6, 2-2
isolatable circuits, 4-6
low-voltage inhibit, 4-14
supply, connecting to, 2-2
supply, requirements, 1-6, 1-8
printed circuit board
description, 4-1
program abort, 1-6, 3-2, 3-14, 3-32, 3-35
prototype area, 1-6, 4-15

—R—

RAM. *See* memory
registers, 2-6, 3-2, 3-11, 3-13, 3-14, 3-15, 3-24, 3-25, 3-26, 3-30, 3-34, 3-35, 4-7, D-1, F-2
reset, 1-6, 2-2, 2-5, 3-1, 3-2, 4-7, 4-14
ROM. *See* memory

—S—

S1, S2. *See* switches
SCI ports
  baud rate, 3-9
  configuration, 2-3, 4-6
  limitations, 3-35
  usage, 1-5, 1-7, 2-3, 2-4
SCI0. *See* SCI ports
SCI1. *See* SCI ports
serial communications interface. *See* SCI ports
Serial Debug Interface (SDI), 1-6, 1-7, 2-4, 4-15, F-1
sockets
  clock oscillator, 4-13
  locations, 1-3
  MCU, 2-1
  memory, 4-9, 4-10
specifications
  EVB, 1-8
speed enhancement, 1-4, 2-6
SRAM. *See* memory
S-Records, 3-17, 3-29, 3-32, A-1
switches, 1-6
  locations, 1-3
  S1 — reset, 3-2
  S2 — program abort, 3-2

—T—

terminal
  baud rate, 2-5, 3-9
  cabling, 2-3, 2-4
  communications parameters, 2-4, 2-5
  communications software, 1-7, 2-5, B-1
  connectors, 2-3, 4-6
  interface circuitry, 4-6
  limitations, 3-35
  requirements, 1-7
  SCI ports, 1-5, 2-3, 4-6
  setup, 2-3, 2-5, 4-6
test points, 1-2, 4-15
time base, 4-14

—U—

upacking instructions, 2-1

—V—

vector memory area, 3-34, F-2

—W—

wait states, 1-4, 2-6, 4-11